BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

CLASS: SE COMPS SEM: IV YEAR: SECOND COURSE CODE: CS205

SUBJECT NAME: DESIGN AND ANALYSIS OF ALGORITHM LAB NO: 604

INDEX

Tools/software used:	**Programming Language:** C, Java, Python **IDE:** Vs Code
Name of the Faculty:	Dr. Sudhir Dhage

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	1		

AIM:	To sort arrays using selection and insertion sort
EXPERIMENT 1	
THEORY:	**Sorting** A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure. In computer science, a sorting algorithm is an algorithm that puts elements of a list into an order. The most frequently used orders are numerical order and lexicographical order, and either ascending or descending. Efficient sorting is important for optimizing the efficiency of other algorithms (such as search and merge algorithms) that require input data to be in sorted lists. Sorting is also often useful for canonicalizing data and for producing human-readable output. Formally, the output of any sorting algorithm must satisfy two conditions: 1. The output is in monotonic order (each element is no smaller/larger than the previous element, according to the required order). 2. The output is a permutation (a reordering, yet retaining all of the original elements) of the input. **Selection sort**

	Selection sort is an in-place comparison sort. It has O(n^2) complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations. The algorithm finds the minimum value, swaps it with the value in the first position, and repeats these steps for the remainder of the list. It does no more than *n* swaps, and thus is useful where swapping is very expensive. **Time Complexity** In computer science, selection sort is an in-place comparison sorting algorithm. It has an O(n^2) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. • Worst case time complexity: Θ(N^2) comparisons and Θ(N) swaps • Average case time complexity: Θ(N^2) comparisons and Θ(N) swaps • Best case time complexity: Θ(N^2) comparisons and Θ(N) swaps • Space complexity: Θ(1) auxillary space **Worst Case Time Complexity** The worst case is the case when the array is already sorted (with one swap) but the smallest element is the last element. For example, if the sorted number as a1, a2, ..., aN, then: a2, a3, ..., aN, a1 will be the worst case for our particular implementation of Selection Sort. Worst Case: a2, a3, ..., aN, a1 The cost in this case is that at each step, a swap is done. This is because the smallest element will always be the last element and the swapped element which is kept at the end will be the second smallest element that is the smallest element of the new unsorted sub-array. Hence, the worst case has:

- N * (N+1) / 2 comparisons
- N swaps

Hence, the time complexity is O(N^2).

Best Case Time Complexity

The best case is the case when the array is already sorted. For example, if the sorted number as a1, a2, ..., aN, then: a1, a2, a3, ..., aN will be the best case for our particular implementation of Selection Sort.
This is the best case as we can avoid the swap at each step but the time spend to find the smallest element is still O(N).
Hence, the best case has:

- N * (N+1) / 2 comparisons
- 0 swaps

Average Case Time Complexity

Based on the worst case and best case, we know that the number of comparisons will be the same for every case and hence, for average case as well, the number of comparisons will be constant.
Number of comparisons = N * (N+1) / 2
Therefore, the time complexity will be O(N^2).
To find the number of swaps,

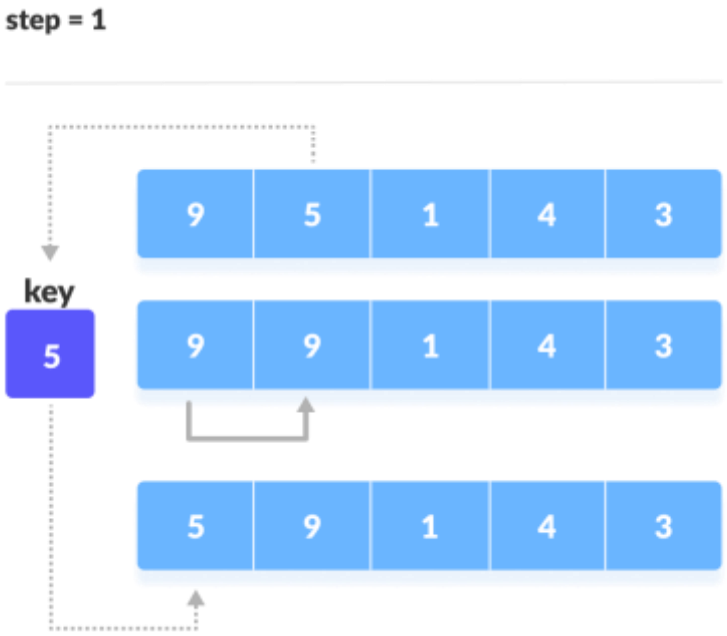
- There are N! different combination of N elements
- Only for one combination (sorted order) there is 0 swaps.
- In the worst case, a combination will have N swaps. There are several such combinations.
- Number of ways to select 2 elements to swap = nC2 = N * (N-1) / 2
- From sorted array, this will result in O(N^2) combinations which need 1 swap.

ALGO:

1. Call insert to insert the element that starts at index 1 into the sorted subarray in index 0.
2. Call insert to insert the element that starts at index 2 into the sorted subarray in indices 0 through 1.

	3. Call insert to insert the element that starts at index 3 into the sorted subarray in indices 0 through 2. 4. … 5. Finally, call insert to insert the element that starts at index n-1*n*−1n, minus, 1 into the sorted subarray in indices 0 through n-2*n*−2n, minus, 2. **Working of Insertion Sort** Suppose we need to sort the following array.  Initial array 1. The first element in the array is assumed to be sorted. Take the second element and store it separately in key. Compare key with the first element. If the first element is greater than key, then key is placed in front of the first

element.



If the first element is greater than key, then key is placed in front of the first element.

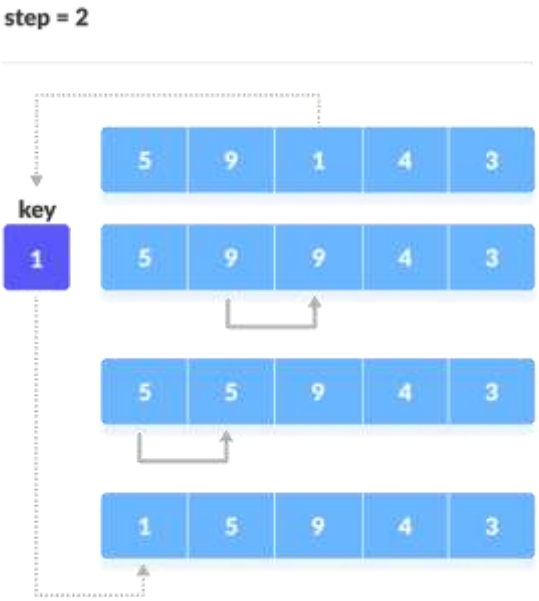
2. Now, the first two elements are sorted.

   Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.

Computer Engineering Department &
Information Technology Engineering Department

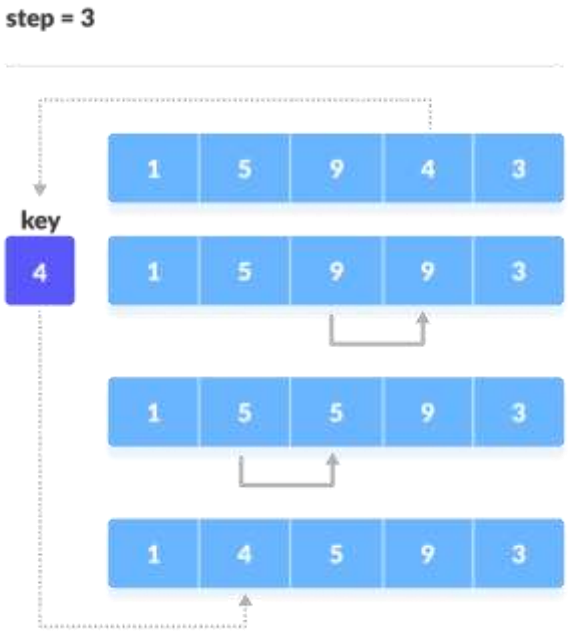
Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

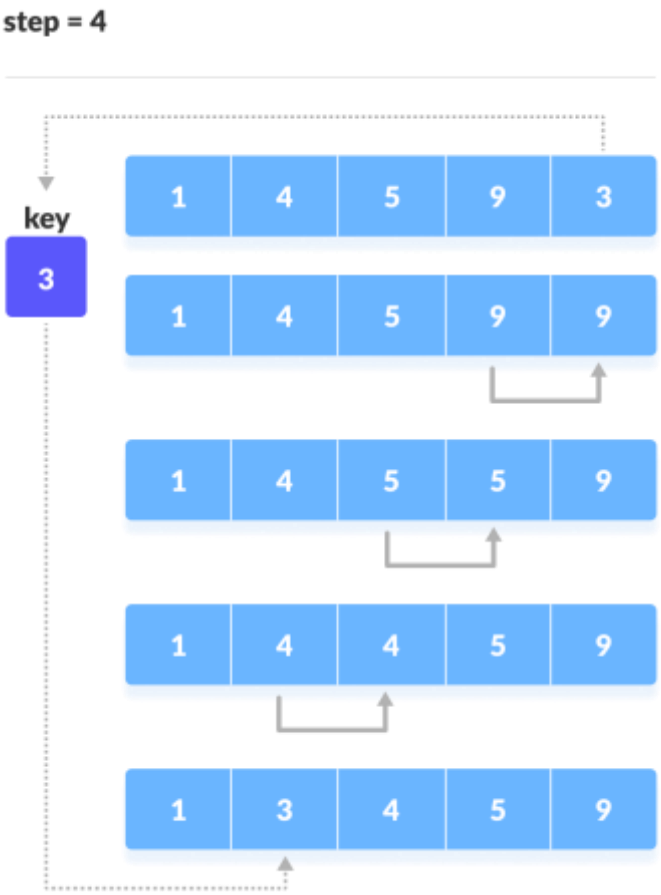


Place 1 at the beginning

3. Similarly, place every unsorted element at its correct position.



Place 4 behind 1



Place 3 behind 1 and the array is sorted

Insertion Sort

Insertion sort is a simple sorting algorithm that is relatively efficient for small lists and mostly sorted lists, and is often used as part of more sophisticated algorithms. It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list similar to how we put money in our wallet. In arrays, the new list and the remaining elements can share the array's space, but insertion is expensive, requiring shifting all following elements over by

one. Shellsort (see below) is a variant of insertion sort that is more efficient for larger lists.

Time Complexity
The worst case time complexity of Insertion sort is O(N^2)
The average case time complexity of Insertion sort is O(N^2)
The time complexity of the best case is O(N).
The space complexity is O(1)

Working Principle

- Compare the element with its adjacent element.
- If at every comparison, we could find a position in sorted array where the element can be inserted, then create space by shifting the elements to right and insert the element at the appropriate position.
- Repeat the above steps until you place the last element of unsorted array to its correct position.

Best Case Analysis
In Best Case i.e., when the array is already sorted, $t_j = 1$
Therefore,$T(n) = C_1 * n + (C_2 + C_3) * (n - 1) + C_4 * (n - 1) + (C_5 + C_6) * (n - 2) + C_8 * (n - 1)$
which when further simplified has dominating factor of n and gives $T(n) = C * (n)$ or $O(n)$

Worst Case Analysis
In Worst Case i.e., when the array is reversly sorted (in descending order), $t_j = j$
Therefore,$T(n) = C_1 * n + (C_2 + C_3) * (n - 1) + C_4 * (n - 1) (n) / 2 + (C_5 + C_6) * ((n - 1) (n) / 2 - 1) + C_8 * (n - 1)$
which when further simplified has dominating factor of n^2 and gives $T(n) = C * (n ^2)$ or $O(n^2)$

Average Case Analysis
Let's assume that $t_j = (j-1)/2$ to calculate the average case
Therefore,$T(n) = C_1 * n + (C_2 + C_3) * (n - 1) + C_4/2 * (n - 1) (n) / 2 + (C_5 + C_6)/2 * ((n - 1) (n) / 2 - 1) + C_8 * ($

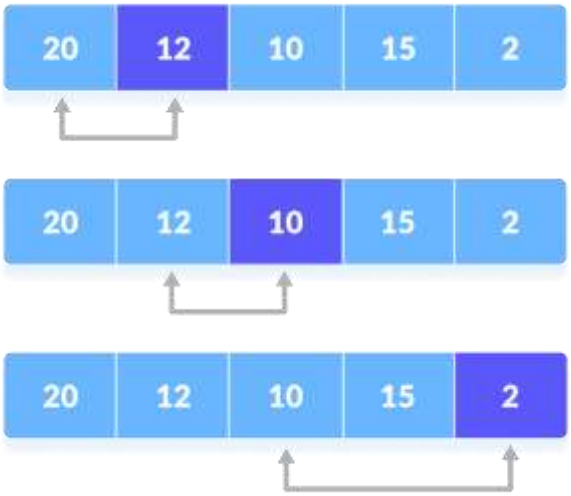
n - 1)
which when further simplified has dominating factor of n^2 and gives $T(n) = C * (n^2)$ or $O(n^2)$

Working of Selection Sort:

1. Set the first element as minimum.

   Select first element as minimum
2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.

   Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.

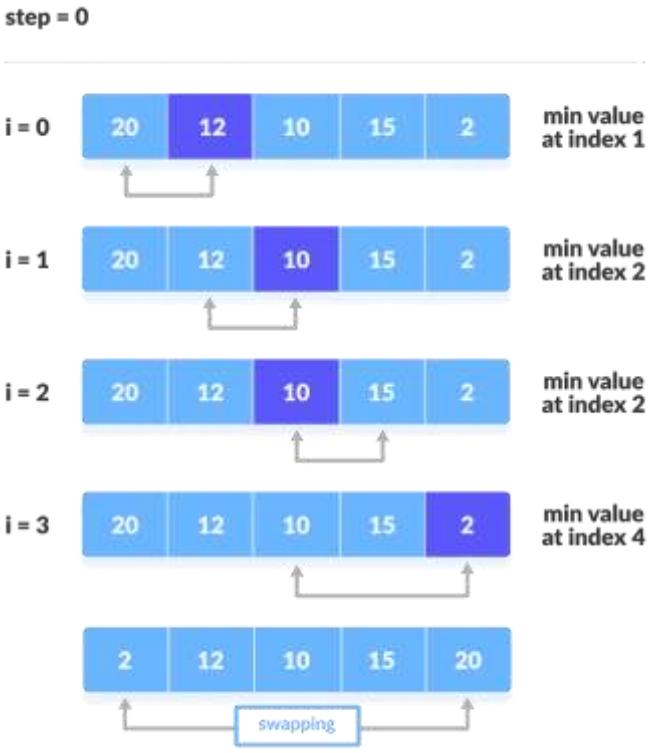


   Compare minimum with the remaining elements

3. After each iteration, minimum is placed in the front of the unsorted list.

Swap the first with minimum

4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

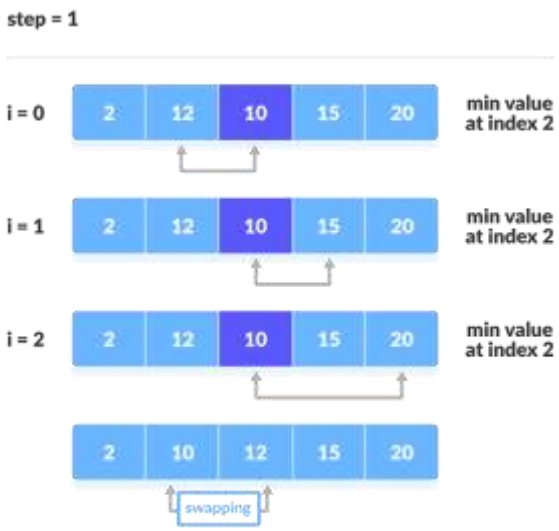


The first iteration

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



The second iteration

The third iteration

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

<table>
<tr><td></td><td>


The fourth iteration

ALGO:
1. Find the smallest card. Swap it with the first card.
2. Find the second-smallest card. Swap it with the second card.
3. Find the third-smallest card. Swap it with the third card.
4. Repeat finding the next-smallest card, and swapping it into the correct position until the array is sorted.
</td></tr>
<tr><td>CODE:</td><td>
<pre>
import java.io.BufferedReader;
import java.io.IOException;
import java.util.*;

public class Sort {
    public static void insertionSort(int array[]) {
        int arrayLen = array.length;
        for (int i = 1; i < arrayLen; ++i) {
            System.out.println("\n**********
**********");
            int currentPos = array[i];
            System.out.print("\nIteration on Element " +
currentPos);
            int j = i - 1;
            while (j >= 0 && array[j] > currentPos) {
</pre>
</td></tr>
</table>

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            System.out.print("\nSwapping " + array[j] + " with
" + array[j + 1]);
            array[j + 1] = array[j];
            j--;
         }
         array[j + 1] = currentPos;
         System.out.print("\nArray : " + printArray(array));
      }
   }

   public static void selectionSort(int array[]) {
      int arrayLen = array.length;

      for (int i = 0; i < arrayLen - 1; i++) {
         System.out.print("\n**********
**********");
         int minElement = i;
         for (int j = i + 1; j < arrayLen; j++) {
            if (array[j] < array[minElement]) {
               minElement = j;
            }
         }
         System.out.print("\nThe minimum Element is " +
array[minElement]);
         System.out.print("\nSwapping " + array[i] + " with "
+ array[minElement]);
         int temp = array[minElement];
         array[minElement] = array[i];
         array[i] = temp;
         System.out.print("\nArray : " + printArray(array));
      }
   }

   public static String printArray(int array[]) {
      String result = "[ ";
      for (int i : array) {
         result += i + " ";
      }
```

```
        result += "]";
        return result;
    }

    public static void main(String[] args) throws IOException {

        ArrayList<Integer> arrayList = new
ArrayList<Integer>();

        int sortingArray[];
        BufferedReader br = new BufferedReader(new
java.io.InputStreamReader(System.in));
        System.out.print("\n1.Manual Input\n2.Random
Input\n3.Roll no Input : ");
        int choice = Integer.parseInt(br.readLine());

        if (choice == 1) {
            System.out.print("\nEnter the elements of the
array(with space)\n-> ");
            String numbers[] = br.readLine().split(" ");
            for (String number : numbers)
                arrayList.add(Integer.parseInt(number));
        } else if (choice == 2) {
            System.out.print("\nEnter the size of the array : ");
            int size = Integer.parseInt(br.readLine());
            for (int i = 0; i < size; i++) {
                arrayList.add((int) (Math.random() * 100));
            }
        } else {
            System.out.print("\nEnter the roll no : ");
            int rollNo = Integer.parseInt(br.readLine());
            for (int i = 0; i < 10; i++)
                arrayList.add(rollNo + (rollNo + 1) * i);
            Collections.shuffle(arrayList);
        }

        sortingArray = new int[arrayList.size()];
        for (int i = 0; i < arrayList.size(); i++) {
```

```
            sortingArray[i] = arrayList.get(i);
        }

        System.out.print("\nArray : " +
printArray(sortingArray));
        System.out.print("\n\n==================
================");
        System.out.print("\nWhich sorting algorithm do you
want to use?\n1. Insertion Sort\n2. Selection Sort\n-> ");
        int decision = Integer.parseInt(br.readLine());
        System.out.print("\n==================
================");
        System.out.print("\nBefore Sort: " +
printArray(sortingArray));
        switch (decision) {
            case 1:
                System.out.print("\nInsertion Sort");
                insertionSort(sortingArray);
                break;
            case 2:
                System.out.print("\nSelection Sort");
                selectionSort(sortingArray);
                break;
            default:
                System.out.print("\nInvalid Choice");
        }

        System.out.print("\n------------------
--------------------------------");
        System.out.print("\nFinal After Sort: " +
printArray(sortingArray));

        System.out.print("\n\n1.Exit\n2.Continue\n-> ");
        int exit = Integer.parseInt(br.readLine());
        if (exit == 1)
            System.exit(0);
        else
            main(null);}}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

OUTPUT:

Insertion Sort:

```
1.Manual Input
2.Random Input
3.Roll no Input : 3

Enter the roll no : 54
```

Best Case:

```
Enter the roll no : 54

1.Best Case
2.Average Case
3.Worst Case : 1

Array : [ 54 109 164 219 274 329 384 439 494 549 ]

================================
Which sorting algorithm do you want to use?
1. Insertion Sort
2. Selection Sort
-> 1

================================
Before Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
Insertion Sort
********************************

Iteration on Element 109
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
********************************

Iteration on Element 164
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
********************************

Iteration on Element 219
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
********************************

Iteration on Element 274
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
********************************
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
Iteration on Element 329
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************

Iteration on Element 384
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************

Iteration on Element 439
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************

Iteration on Element 494
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************

Iteration on Element 549
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
------------------------------------------------------
Final After Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
```

Average Case:

```
1.Best Case
2.Average Case
3.Worst Case : 2

Array : [ 109 274 54 384 164 494 439 549 329 219 ]

================================
Which sorting algorithm do you want to use?
1. Insertion Sort
2. Selection Sort
-> 1

================================
Before Sort: [ 109 274 54 384 164 494 439 549 329 219 ]
Insertion Sort
********************************

Iteration on Element 274
Array : [ 109 274 54 384 164 494 439 549 329 219 ]
********************************
```

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
Iteration on Element 54
Swapping 274 with 54
Swapping 109 with 274
Array : [ 54 109 274 384 164 494 439 549 329 219 ]
**********************************

Iteration on Element 384
Array : [ 54 109 274 384 164 494 439 549 329 219 ]
**********************************

Iteration on Element 164
Swapping 384 with 164
Swapping 274 with 384
Array : [ 54 109 164 274 384 494 439 549 329 219 ]
**********************************

Iteration on Element 494
Array : [ 54 109 164 274 384 494 439 549 329 219 ]
**********************************

Iteration on Element 439
Swapping 494 with 439
Array : [ 54 109 164 274 384 439 494 549 329 219 ]
**********************************

Iteration on Element 549
Array : [ 54 109 164 274 384 439 494 549 329 219 ]
**********************************

Iteration on Element 329
Swapping 549 with 329
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Array : [ 54 109 164 274 329 384 439 494 549 219 ]
**********************************

Iteration on Element 219
Swapping 549 with 219
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Swapping 329 with 384
Swapping 274 with 329
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
--------------------------------------------------
Final After Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Worth Case:

```
================================
Before Sort: [ 549 494 439 384 329 274 219 164 109 54 ]
Insertion Sort
********************************

Iteration on Element 494
Swapping 549 with 494
Array : [ 494 549 439 384 329 274 219 164 109 54 ]
********************************

Iteration on Element 439
Swapping 549 with 439
Swapping 494 with 549
Array : [ 439 494 549 384 329 274 219 164 109 54 ]
********************************

Iteration on Element 384
Swapping 549 with 384
Swapping 494 with 549
Swapping 439 with 494
Array : [ 384 439 494 549 329 274 219 164 109 54 ]
********************************

Iteration on Element 329
Swapping 549 with 329
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Array : [ 329 384 439 494 549 274 219 164 109 54 ]
********************************

Iteration on Element 274
Swapping 549 with 274
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Swapping 329 with 384
Array : [ 274 329 384 439 494 549 219 164 109 54 ]
********************************
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

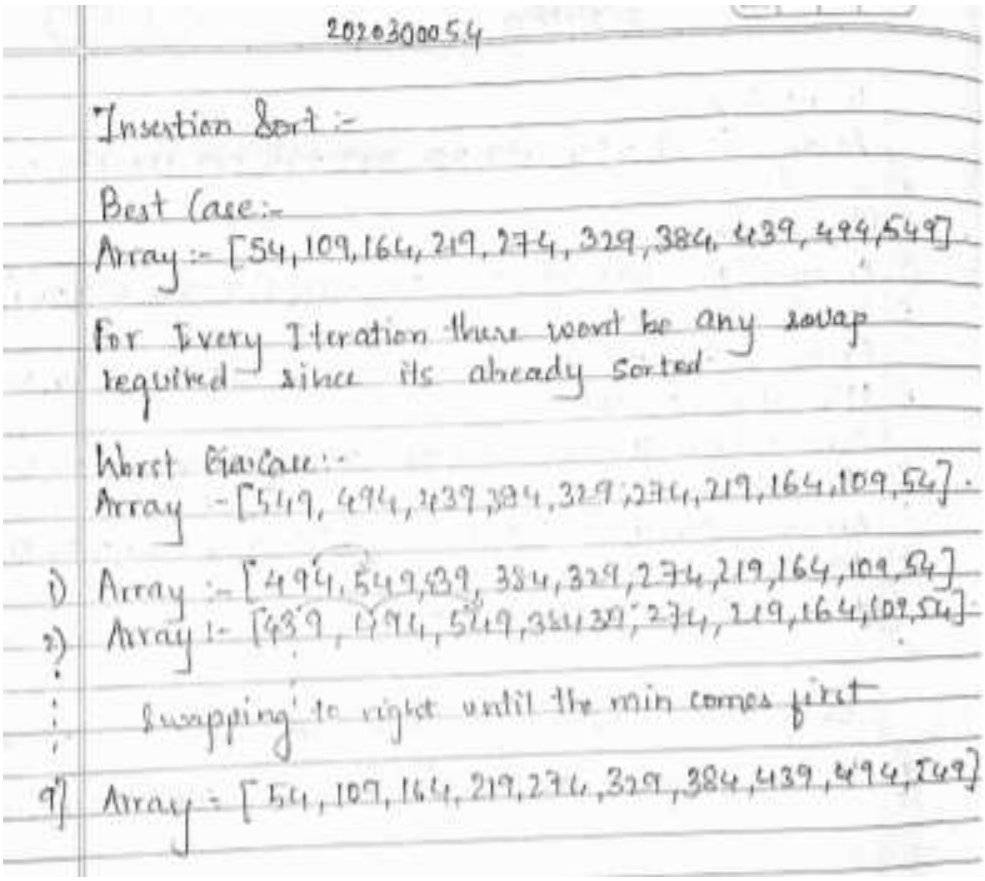
```
Iteration on Element 219
Swapping 549 with 219
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Swapping 329 with 384
Swapping 274 with 329
Array : [ 219 274 329 384 439 494 549 164 109 54 ]
**********************************

Iteration on Element 164
Swapping 549 with 164
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Swapping 329 with 384
Swapping 274 with 329
Swapping 219 with 274
Array : [ 164 219 274 329 384 439 494 549 109 54 ]
**********************************

Iteration on Element 109
Swapping 549 with 109
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Swapping 329 with 384
Swapping 274 with 329
Swapping 219 with 274
Swapping 164 with 219
Array : [ 109 164 219 274 329 384 439 494 549 54 ]
**********************************

Iteration on Element 54
Swapping 549 with 54
Swapping 494 with 549
Swapping 439 with 494
Swapping 384 with 439
Swapping 329 with 384
Swapping 274 with 329
Swapping 219 with 274
Swapping 164 with 219
Swapping 109 with 164
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
------------------------------------------------------
Final After Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



For the Average Case:
Every Iteration, the array element will float up using swaps

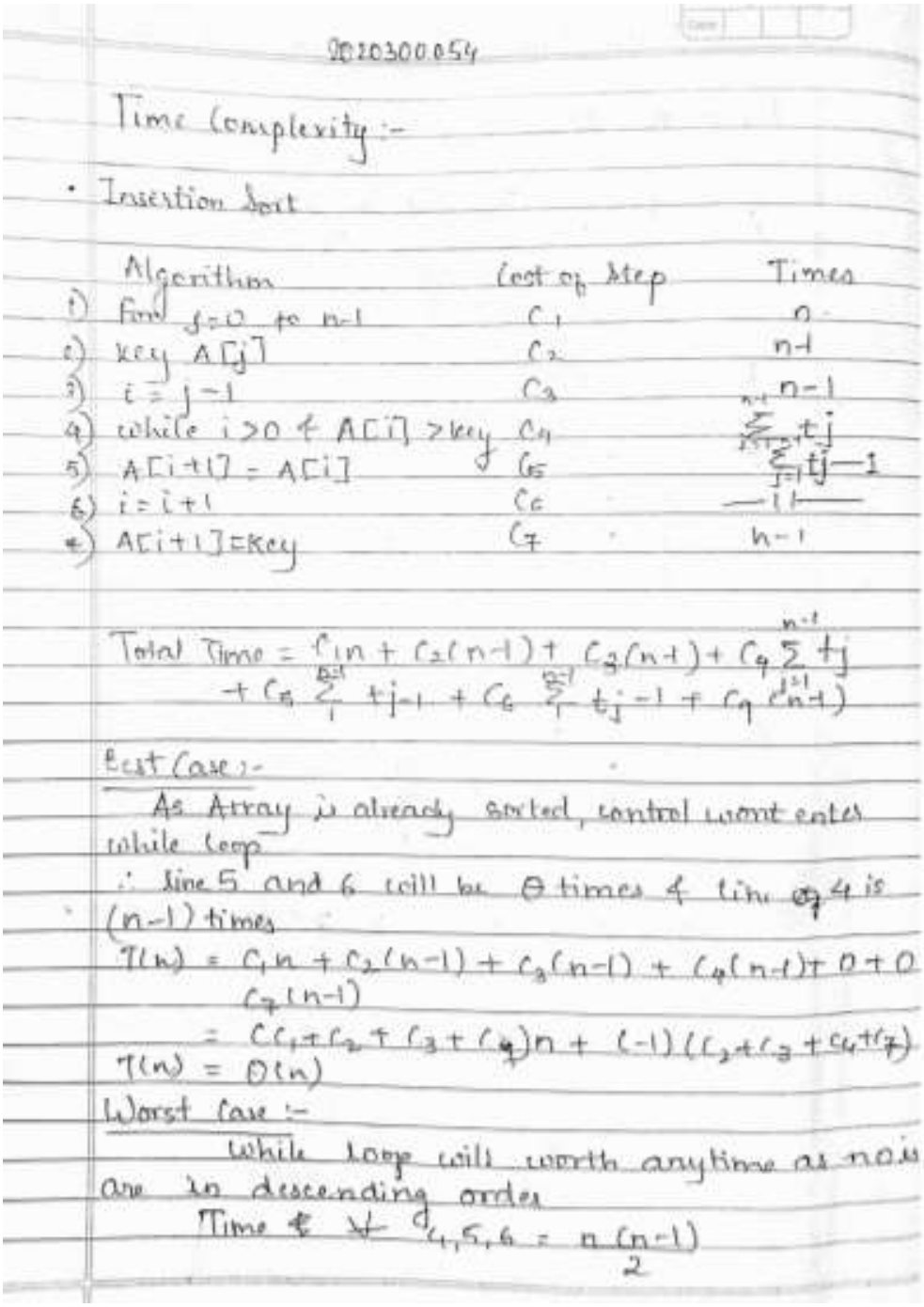
Complexity of Insertion Sort: The best case complexity of insertion sort is O(n) times, i.e. when the array is previously sorted. In the same way, when the array is sorted in reverse order, the first element of the unsorted array is to be compared with each element in the sorted set. So, in the worst case,running time of Insertion sort is quadratic, i.e., O(n 2).

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



2020300054

Time Complexity :-

- Insertion Sort

Algorithm	Cost of Step	Times
1) Find $j=0$ to $n-1$	c_1	n
2) key $A[j]$	c_2	$n-1$
3) $i = j-1$	c_3	$n-1$
4) while $i>0$ & $A[i] >$ key	c_4	$\sum_{j=1}^{n-1} t_j$
5) $A[i+1] = A[i]$	c_5	$\sum_{j=1}^{n-1} t_j - 1$
6) $i = i+1$	c_6	$\sum_{j=1}^{n-1} t_j - 1$
7) $A[i+1] =$ key	c_7	$n-1$

Total Time $= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=1}^{n-1} t_j + c_5 \sum_{j=1}^{n-1} t_j - 1 + c_6 \sum_{j=1}^{n-1} t_j - 1 + c_7 (n-1)$

Best Case :-

As Array is already sorted, control wont enter while loop

∴ line 5 and 6 will be 0 times & line 4 is $(n-1)$ times

$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + 0 + 0 + c_7(n-1)$

$= (c_1 + c_2 + c_3 + c_4)n + (-1)(c_2 + c_3 + c_4 + c_7)$

$T(n) = O(n)$

Worst Case :-

While loop will worth anytime as nos are in descending order

Time of $4,5,6 = \frac{n(n-1)}{2}$

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Selection Sort:

Best Case:

```
================================
Before Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
Selection Sort
**********************************
The minimum Element is 54
Swapping 54 with 54
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 109
Swapping 109 with 109
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 164
Swapping 164 with 164
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 219
Swapping 219 with 219
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 274
Swapping 274 with 274
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 329
Swapping 329 with 329
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 384
Swapping 384 with 384
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 439
Swapping 439 with 439
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 494
Swapping 494 with 494
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
------------------------------------------------
Final After Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Average Case:

```
================================
Before Sort: [ 329 494 164 54 439 219 384 109 549 274 ]
Selection Sort
********************************
The minimum Element is 54
Swapping 329 with 54
Array : [ 54 494 164 329 439 219 384 109 549 274 ]
********************************
The minimum Element is 109
Swapping 494 with 109
Array : [ 54 109 164 329 439 219 384 494 549 274 ]
********************************
The minimum Element is 164
Swapping 164 with 164
Array : [ 54 109 164 329 439 219 384 494 549 274 ]
********************************
The minimum Element is 219
Swapping 329 with 219
Array : [ 54 109 164 219 439 329 384 494 549 274 ]
********************************
The minimum Element is 274
Swapping 439 with 274
Array : [ 54 109 164 219 274 329 384 494 549 439 ]
********************************
The minimum Element is 329
Swapping 329 with 329
Array : [ 54 109 164 219 274 329 384 494 549 439 ]
********************************
The minimum Element is 384
Swapping 384 with 384
Array : [ 54 109 164 219 274 329 384 494 549 439 ]
********************************
The minimum Element is 439
Swapping 494 with 439
Array : [ 54 109 164 219 274 329 384 439 549 494 ]
********************************
The minimum Element is 494
Swapping 549 with 494
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
--------------------------------------------------
Final After Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Worst Case:

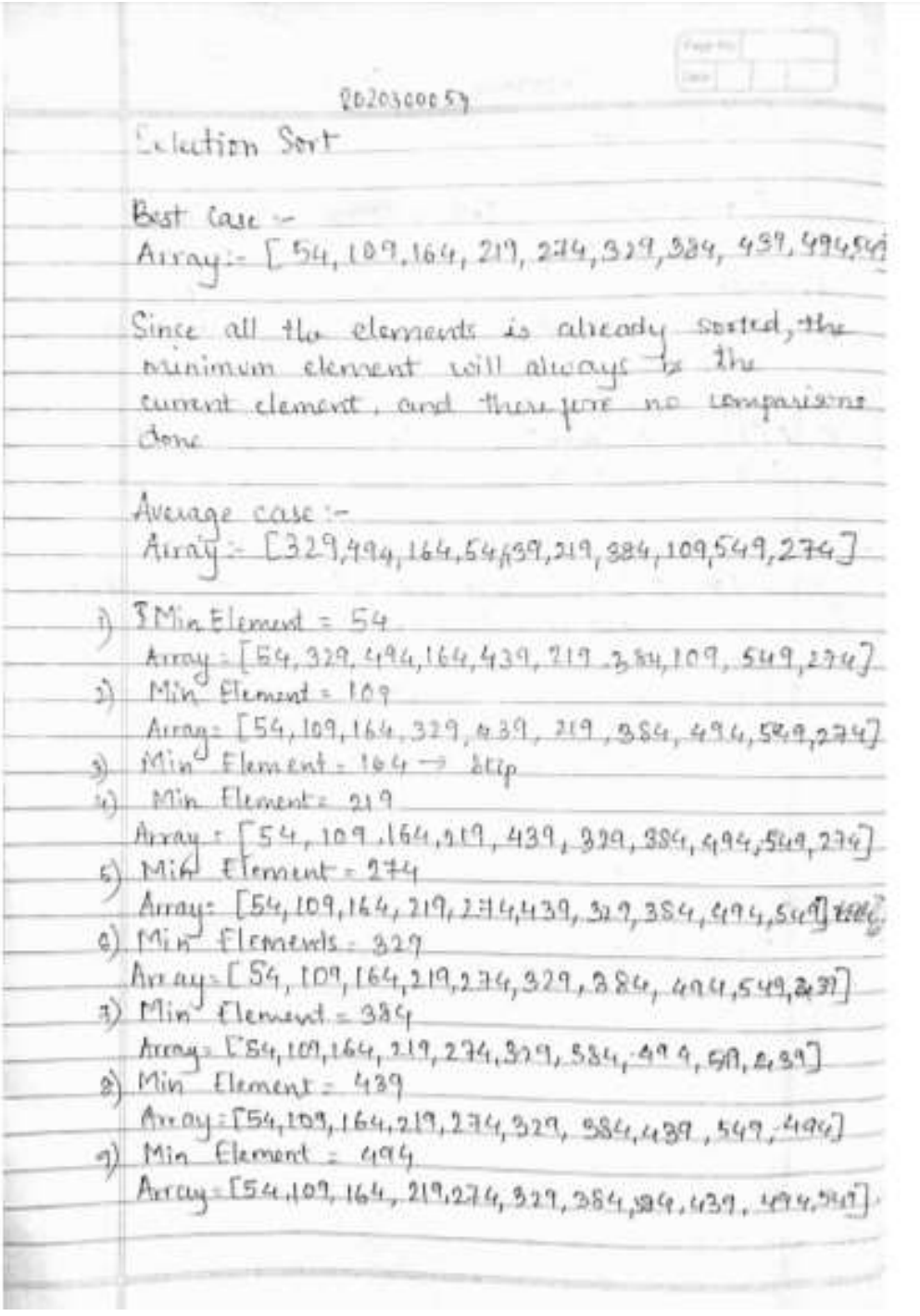
```
==================================
Before Sort: [ 549 494 439 384 329 274 219 164 109 54 ]
Selection Sort
**********************************
The minimum Element is 54
Swapping 549 with 54
Array : [ 54 494 439 384 329 274 219 164 109 549 ]
**********************************
The minimum Element is 109
Swapping 494 with 109
Array : [ 54 109 439 384 329 274 219 164 494 549 ]
**********************************
The minimum Element is 164
Swapping 439 with 164
Array : [ 54 109 164 384 329 274 219 439 494 549 ]
**********************************
The minimum Element is 219
Swapping 384 with 219
Array : [ 54 109 164 219 329 274 384 439 494 549 ]
**********************************
The minimum Element is 274
Swapping 329 with 274
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 329
Swapping 329 with 329
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 384
Swapping 384 with 384
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 439
Swapping 439 with 439
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
**********************************
The minimum Element is 494
Swapping 494 with 494
Array : [ 54 109 164 219 274 329 384 439 494 549 ]
----------------------------------------------------
Final After Sort: [ 54 109 164 219 274 329 384 439 494 549 ]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

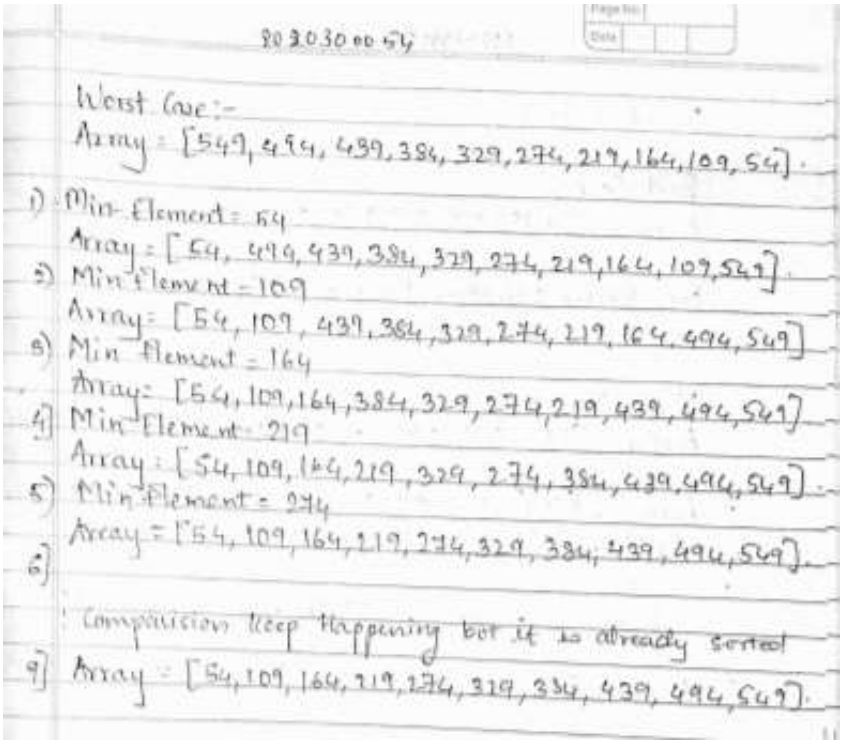


Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Selection Sort

Algorithm	Cost of Steep	Times
1) For i=0 to A.length	c_1	n
2) min = i	c_2	$n-1$
3) for j=i+1 to A.len	c_3	$\frac{n(n+1)}{2}$
4) temp = A[i]	c_4	$n-1$
5) A[i] = A[min]	c_5	$n-1$
6) A[min] = temp	c_6	$n-1$

Total Time

$T(n) = c_1 n + c_2(n-1) + c_3 \frac{n(n-1)}{2} + c_4(n-1) + c_5(n-1) + c_6(n-1)$

$= \frac{c_3}{2} n^2 + \left(c_1 + c_2 + \frac{c_3}{2} + c_4 + c_5 + c_6\right) n - (c_2 + c_4 + c_5 + c_6)$

$= an^2 + bn + c$

$= O(n^2)$

Time complexity = $O(n^2)$ for all cases

Complexity of Selection Sort: As the working of selection,sort does not depend on the original order of the elements in the array,so there is not much difference between best case and worst case complexity of selection sort. Therefore, in both the cases, the complexity is O(n 2).

CONCLUSION: Concepts learnt during procedural programming of the problem

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

- Learnt about the time complexity and proved and perform complexity cases on selection and insertion sort
- Learnt how to write Insertion and Selection sort using the algorithm
- Learnt how to different cases differ in computation power as the number of elements increase in array
- I learnt about the advantages and disadvantages about selection and insertion sort and how to improve the overall time complexity
- Learnt how to derive and compute the time complexity of selection sort and insertion sort

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	2		

AIM:	To implement divide and conquer sorting algorithms
THEORY:	**What is QuickSort ?** Quick Sort is a divide and conquer algorithm. It creates two empty arrays to hold elements less than the pivot value and elements greater than the pivot value, and then recursively sort the sub arrays. There are two basic operations in the algorithm, swapping items in place and partitioning a section of the array. **Important Characteristics of Quick Sort:** • Quick Sort is useful for sorting arrays. • In efficient implementations Quick Sort is not a stable sort, meaning that the relative order of equal sort items is not preserved. • Overall time complexity of Quick Sort is O(nLogn). In the worst case, it makes O(*n*2) comparisons, though this behavior is rare. • The space complexity of Quick Sort is O(nLogn). It is an in-place sort (i.e. it doesn't require any extra storage) **Problem Statement** Consider the following array: 50, 23, 9, 18, 61, 32. We need to sort this array in the most efficient manner without using extra place (inplace sorting).

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Solution

Step 1:

- **Make any element as pivot:** Decide any value to be the pivot from the list. For convenience of code, we often select the rightmost index as pivot or select any at random and swap with rightmost. Suppose for two values "Low" and "High" corresponding to the first index and last index respectively.
  - In our case low is 0 and high is 5.
  - Values at low and high are 50 and 32 and value at pivot is 32.
- **Partition the array on the basis of pivot:** Call for partitioning which rearranges the array in such a way that pivot (32) comes to its actual position (of the sorted array). And to the left of the pivot, the array has all the elements less than it, and to the right greater than it.
  - In the partition function, we start from the first element and compare it with the pivot. Since 50 is greater than 32, we don't make any change and move on to the next element 23.
  - Compare again with the pivot. Since 23 is less than 32, we swap 50 and 23. The array becomes 23, 50, 9, 18, 61, 32
  - We move on to the next element 9 which is again less than pivot (32) thus swapping it with 50 makes our array as 23, 9, 50, 18, 61, 32.
  - Similarly, for next element 18 which is less than 32, the array becomes 23, 9, 18, 50, 61, 32. Now 61 is greater than pivot (32), hence no changes.
  - Lastly, we swap our pivot with 50 so that it comes to the correct position.

Thus the pivot (32) comes at its actual position and all elements to its left are lesser, and all elements to the right are greater than itself.

Step 2: The main array after the first step becomes
23, 9, 18, 32, 61, 50

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

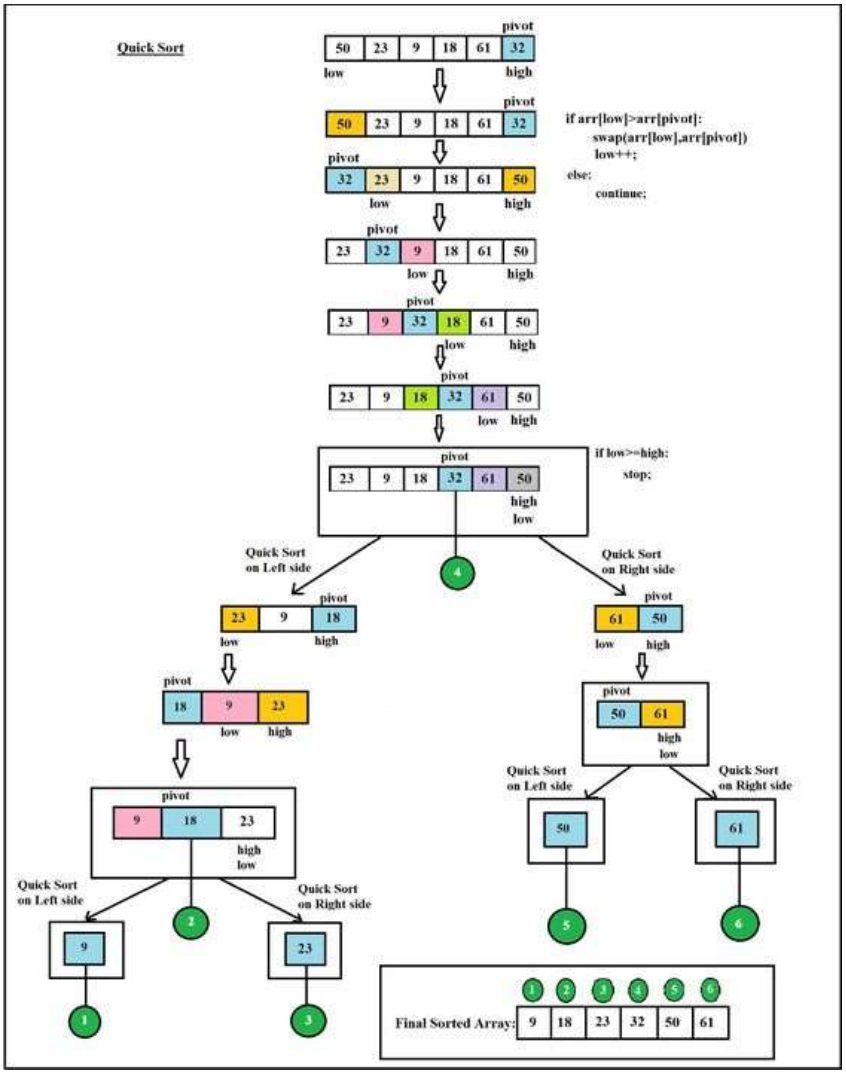
Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Step 3: Now the list is divided into two parts:

1. Sublist before pivot element
2. Sublist after pivot element

Step 4: Repeat the steps for the left and right sublists recursively. The final array thus becomes
9, 18, 23, 32, 50, 61.

The following diagram depicts the workflow of the Quick Sort algorithm which was described above.



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

ALGORITHM:

1. Find a "pivot" item in the array. This item is the basis for comparison for a single round.
2. Start a pointer (the left pointer) at the first item in the array.
3. Start a pointer (the right pointer) at the last item in the array.
4. While the value at the left pointer in the array is less than the pivot value, move the left pointer to the right (add 1). Continue until the value at the left pointer is greater than or equal to the pivot value.
5. While the value at the right pointer in the array is greater than the pivot value, move the right pointer to the left (subtract 1). Continue until the value at the right pointer is less than or equal to the pivot value.
6. If the left pointer is less than or equal to the right pointer, then swap the values at these locations in the array.
7. Move the left pointer to the right by one and the right pointer to the left by one.
8. If the left pointer and right pointer don't meet, go to step 1

Appliations of quick sort

- Commercial Computing is used in various government and private organizations for the purpose of sorting various data like sorting files by name/date/price, sorting of students by their roll no., sorting of account profile by given id, etc.
- The sorting algorithm is used for information searching and as Quicksort is the fastest algorithm so it is widely used as a better way of searching.
- It is used everywhere where a stable sort is not needed.
- It is tail -recursive and hence all the call optimization can be done.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	
CODE:	Program Code Quicksort: import java.util.Arrays; public class quickSort { int totalSwaps = 0; int totalComparisons = 0; *// This function takes last element as pivot, places* public int partition(int arr[], int low, int high) { int pivot = low; low++; int comparisons = 0; int swaps = 0; System.out.print("\n---------- ----------"); System.out.print("\nPivot: " + arr[high] + " Low: " + low + " High: " + high); System.out.print("\nBefore Swaps-> Array: " + printArray(arr)); do { *// increment low pointer until that element is larger than the pivot element* while (low < high && arr[low] <= arr[pivot]) { low++; comparisons++; } *// decrement high pointer until that element is smaller than the pivot element* while (high > pivot && arr[high] > arr[pivot]) { high--; comparisons++; }

```
            // if the low and high pointers cross each other swap
the corresponding elements
            if (low < high) {
                int temp = arr[low];
                arr[low] = arr[high];
                arr[high] = temp;
                swaps++;

            }
        } while (low < high);
        // swap the pivot element and the element pointed by
high
        System.out.print("\n\nSwapping pivot and high: " +
arr[pivot] + " with " + arr[high]);
        int temp = arr[high];
        arr[high] = arr[pivot];
        arr[pivot] = temp;
        totalComparisons += comparisons;
        totalSwaps += swaps;
        System.out.println("\n\nAfter Swaps -> Array: " +
printArray(arr));
        System.out.print("\nSwaps: " + swaps + "
Comparisons: " + comparisons);
        return high;
    }

    // swap function
    public void swap(int arr[], int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public void sort(int arr[], int low, int high) {
        if (low < high) {
            /*
             * pi is partitioning index, arr[pi] is
             * now at right place
             */
```

```
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }

    /* A utility function to print array of size n */
    public String printArray(int arr[]) {
        return Arrays.toString(arr);
    }
}

Main Class:
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class Driver {
    public static void main(String[] args) {
        // User Input
        Scanner input = new Scanner(System.in);
        System.out.print("\n                          QUICKSORT");
        System.out.print("\n----------------------------------------
---------------------------\n");
        System.out.print("\nEnter the roll no: ");
        int rollNo = input.nextInt();

        int array[];
        ArrayList<Integer> list = new ArrayList<Integer>();

        // Case input
        System.out.print("\n1.Random Case\n2.Worst
Case\n3.Manual Case  :");
        int choice = input.nextInt();
        if (choice == 2) {
            // Worst Case
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            for (int i = 9; i >= 0; i--)
                list.add(rollNo + (rollNo + 1) * i);

        } else if (choice == 1) {
            // Random Case
            for (int i = 0; i < 10; i++)
                list.add(rollNo + (rollNo + 1) * i);

            Collections.shuffle(list);
        } else {
            // Manual Case
            System.out.print("\nEnter the elements: ");
            for (int i = 0; i < 10; i++)
                list.add(input.nextInt());
        }
        array = new int[10];
        for (int i = 0; i < 10; i++)
            array[i] = list.get(i);
        quickSort qSort = new quickSort();
        System.out.print("\nBefore Sorting: " +
qSort.printArray(array));
        System.out.print("\n----------------------------------------
---------------------------\n");
        qSort.sort(array, 0, array.length - 1);
        System.out.print("\n----------------------------------------
---------------------------\n");
        System.out.print("\nAfter Sorting: " +
qSort.printArray(array));
        System.out.print("\n----------------------------------------
---------------------------\n");
        System.out.print("\nTotal swaps: " +
qSort.totalSwaps);
        System.out.print("\nTotal comparisons: " +
qSort.totalComparisons);
        input.close();
    }
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

OUTPUT:

Best case

```
                              QUICKSORT
-----------------------------------------------------------------

Enter the roll no: 54

1.Random Case
2.Worst Case
3.Manual Case  :3

Enter the elements: 274 54 109 219 164 439 329 384 549 494

Before Sorting: [274, 54, 109, 219, 164, 439, 329, 384, 549, 494]
-----------------------------------------------------------------

-------------------------------------------------------------
Pivot: 494 Low: 1 High: 9
Before Swaps-> Array: [274, 54, 109, 219, 164, 439, 329, 384, 549, 494]

Swapping pivot and high: 274 with 164

After Swaps -> Array: [164, 54, 109, 219, 274, 439, 329, 384, 549, 494]

Swaps: 0 Comparisons: 9
-------------------------------------------------------------
Pivot: 219 Low: 1 High: 3
Before Swaps-> Array: [164, 54, 109, 219, 274, 439, 329, 384, 549, 494]

Swapping pivot and high: 164 with 109

After Swaps -> Array: [109, 54, 164, 219, 274, 439, 329, 384, 549, 494]
Swaps: 0 Comparisons: 3
-------------------------------------------------------------
Pivot: 54 Low: 1 High: 1
Before Swaps-> Array: [109, 54, 164, 219, 274, 439, 329, 384, 549, 494]

Swapping pivot and high: 109 with 54

After Swaps -> Array: [54, 109, 164, 219, 274, 439, 329, 384, 549, 494]
Swaps: 0 Comparisons: 0
-------------------------------------------------------------
Pivot: 494 Low: 6 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 439, 329, 384, 549, 494]

Swapping pivot and high: 439 with 384

After Swaps -> Array: [54, 109, 164, 219, 274, 384, 329, 439, 549, 494]
Swaps: 0 Comparisons: 4
```

```
------------------------------------------------------------
Pivot: 329 Low: 6 High: 6
Before Swaps-> Array: [54, 109, 164, 219, 274, 384, 329, 439, 549, 494]

Swapping pivot and high: 384 with 329

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]
Swaps: 0 Comparisons: 0
------------------------------------------------------------
Pivot: 494 Low: 9 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]

Swapping pivot and high: 549 with 494

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
Swaps: 0 Comparisons: 0
------------------------------------------------------------------

After Sorting: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
------------------------------------------------------------------

Total swaps: 0
Total comparisons: 16
```

Time Complexity of Quick sort

- **Best case scenario:** The best case scenario occurs when the partitions are as evenly balanced as possible, i.e their sizes on either side of the pivot element are either are equal or are have size difference of 1 of each other.
    - Case 1: The case when sizes of sublist on either side of pivot becomes equal occurs when the subarray has an odd number of elements and the pivot is right in the middle after partitioning. Each partition will have (n-1)/2 elements.
    - Case 2: The size difference of 1 between the two sublists on either side of pivot happens if the subarray has an even number, n, of elements. One partition will have n/2 elements with the other having (n/2)-1.

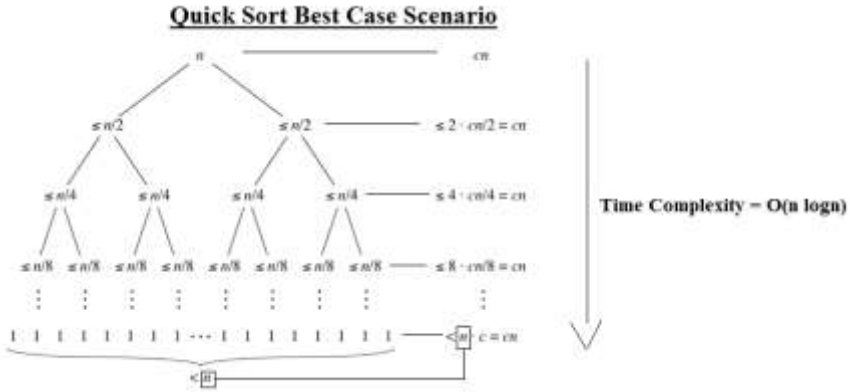
In either of these cases, each partition will have at most n/2 elements, and the tree representation of the subproblem sizes will be as below:

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

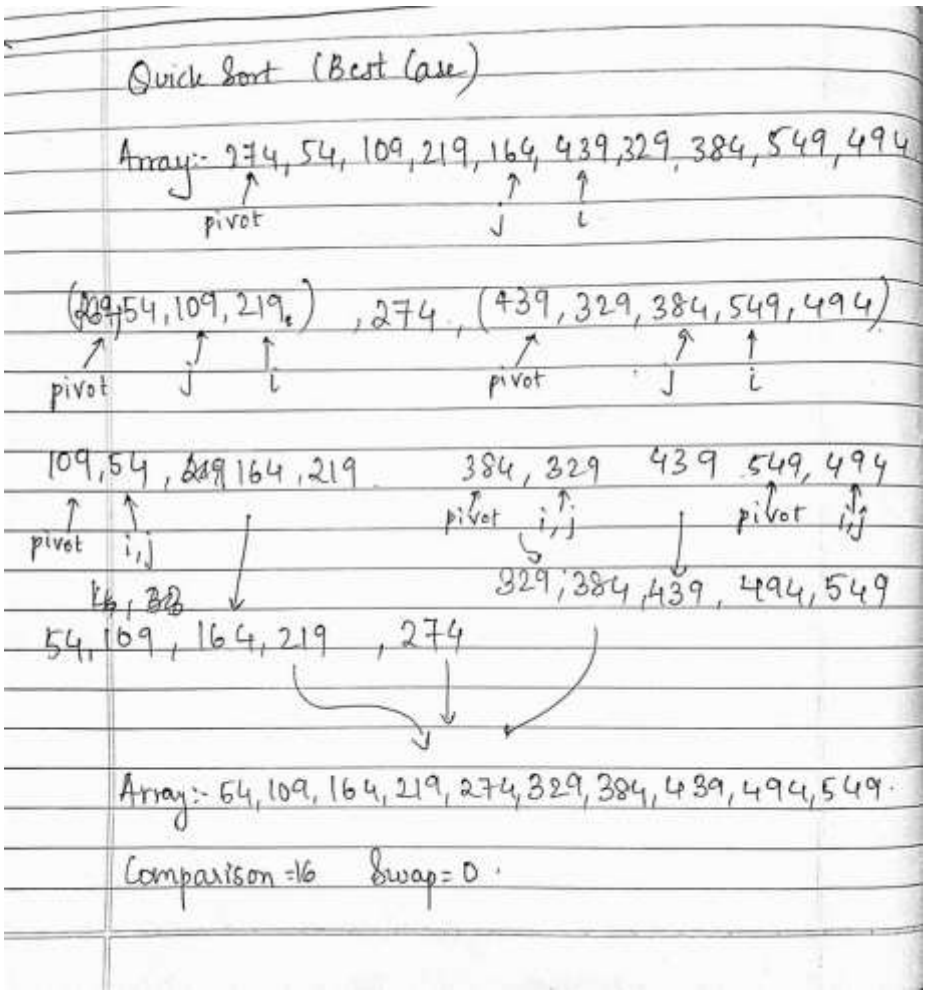
Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



The best-case complexity of the quick sort algorithm is O(n logn)



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Worst Case:

```
Before Sorting: [549, 494, 439, 384, 329, 274, 219, 164, 109, 54]
----------------------------------------------------------------


------------------------------------------------------------
Pivot: 54 Low: 1 High: 9
Before Swaps-> Array: [549, 494, 439, 384, 329, 274, 219, 164, 109, 54]

Swapping pivot and high: 549 with 54

After Swaps -> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]
Swaps: 0 Comparisons: 8
------------------------------------------------------------
Pivot: 109 Low: 1 High: 8
Before Swaps-> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]

Swapping pivot and high: 54 with 54

After Swaps -> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]
Swaps: 0 Comparisons: 8
------------------------------------------------------------
Pivot: 109 Low: 2 High: 8
Before Swaps-> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]

Swapping pivot and high: 494 with 109

After Swaps -> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]
Swaps: 0 Comparisons: 6
------------------------------------------------------------
Pivot: 164 Low: 2 High: 7
Before Swaps-> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]

Swapping pivot and high: 109 with 109

After Swaps -> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]
Swaps: 0 Comparisons: 6
------------------------------------------------------------
```

```
------------------------------------------------------------
Pivot: 164 Low: 3 High: 7
Before Swaps-> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]

Swapping pivot and high: 439 with 164

After Swaps -> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]
Swaps: 0 Comparisons: 4
------------------------------------------------------------
Pivot: 219 Low: 3 High: 6
Before Swaps-> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]

Swapping pivot and high: 164 with 164

After Swaps -> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]
Swaps: 0 Comparisons: 4
------------------------------------------------------------
Pivot: 219 Low: 4 High: 6
Before Swaps-> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]

Swapping pivot and high: 384 with 219

After Swaps -> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]
Swaps: 0 Comparisons: 2
```

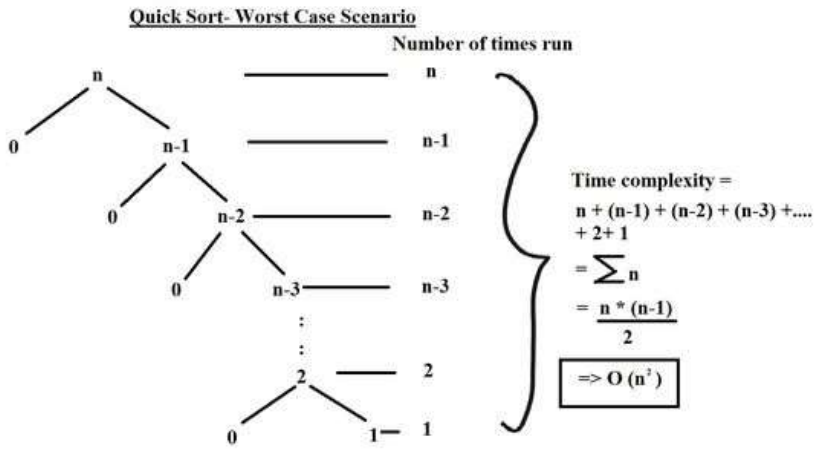
Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
------------------------------------------------------------
Pivot: 274 Low: 4 High: 5
Before Swaps-> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]

Swapping pivot and high: 219 with 219

After Swaps -> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]
Swaps: 0 Comparisons: 2
------------------------------------------------------------
Pivot: 274 Low: 5 High: 5
Before Swaps-> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]

Swapping pivot and high: 329 with 274

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
Swaps: 0 Comparisons: 0
------------------------------------------------------------------

After Sorting: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
------------------------------------------------------------------

Total swaps: 0
Total comparisons: 40
```

- **Worst case scenario:** This happens when we encounter the most unbalanced partitions possible, then the original call takes n time, the recursive call on n-1 elements will take (n-1) time, the recursive call on (n-2) elements will take (n-2) time, and so on. The worst case time complexity of Quick Sort would be **O(n2)**.

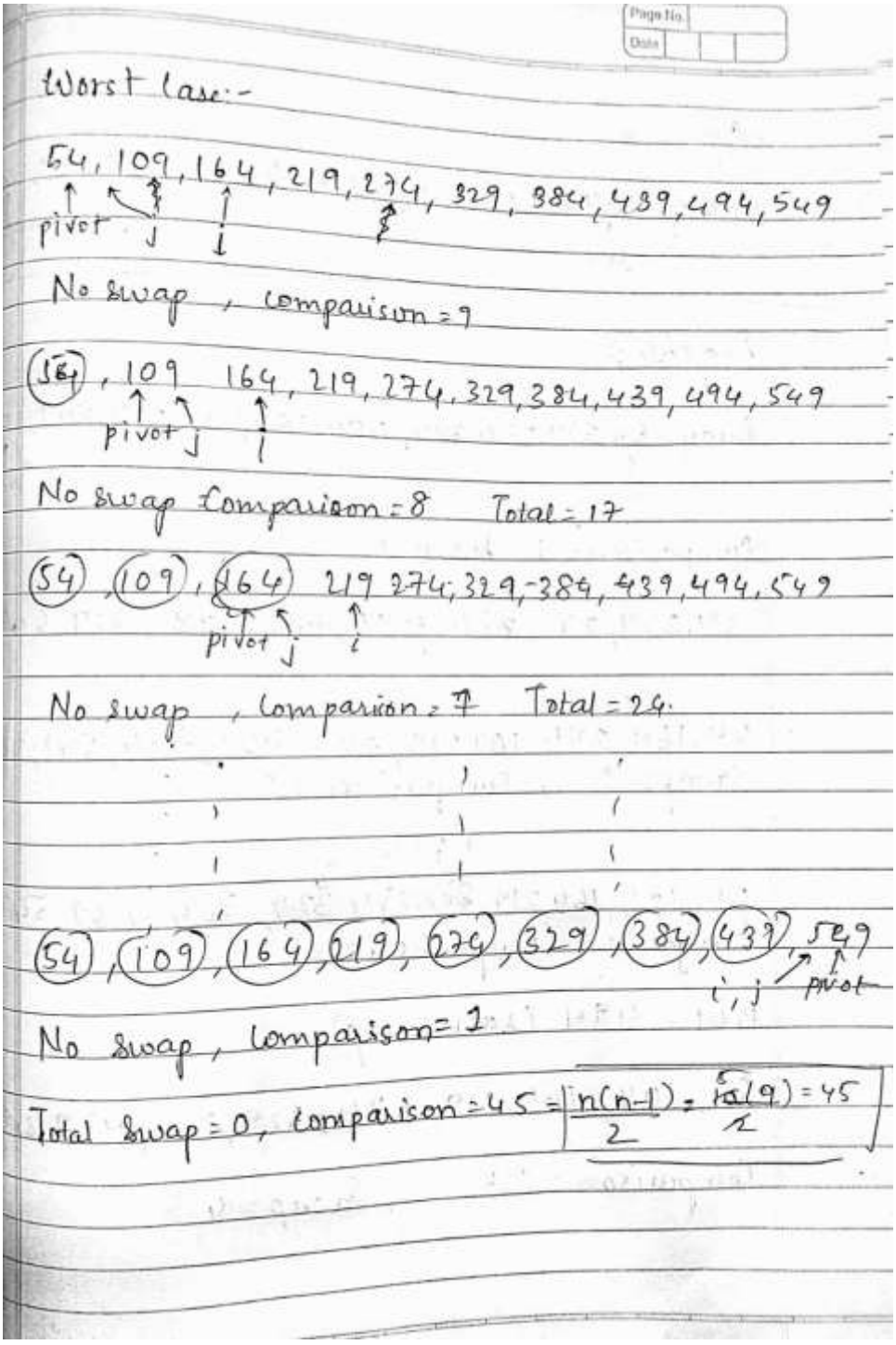


Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

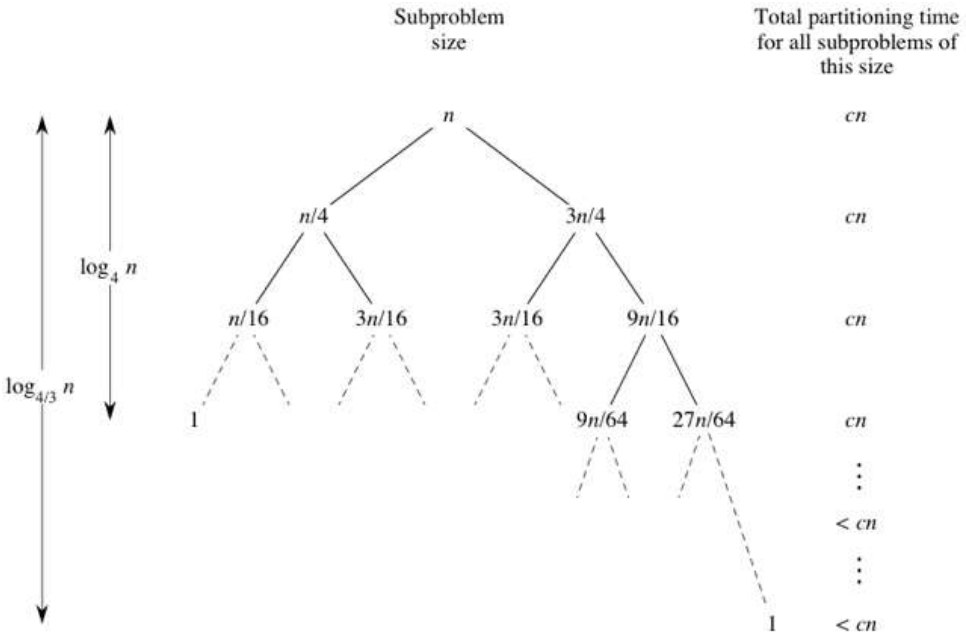
Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Average Case:

```
Before Sorting: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]
--------------------------------------------------------------------


------------------------------------------------------------
Pivot: 109 Low: 1 High: 9
Before Swaps-> Array: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]

Swapping pivot and high: 54 with 54

After Swaps -> Array: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]
Swaps: 0 Comparisons: 9
------------------------------------------------------------
Pivot: 109 Low: 2 High: 9
Before Swaps-> Array: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]

Swapping pivot and high: 329 with 164

After Swaps -> Array: [54, 164, 274, 109, 219, 329, 439, 494, 549, 384]
Swaps: 2 Comparisons: 8
------------------------------------------------------------
Pivot: 219 Low: 2 High: 4
Before Swaps-> Array: [54, 164, 274, 109, 219, 329, 439, 494, 549, 384]

Swapping pivot and high: 164 with 109

After Swaps -> Array: [54, 109, 164, 274, 219, 329, 439, 494, 549, 384]
Swaps: 1 Comparisons: 3
------------------------------------------------------------
Pivot: 219 Low: 4 High: 4
Before Swaps-> Array: [54, 109, 164, 274, 219, 329, 439, 494, 549, 384]

Swapping pivot and high: 274 with 219

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 439, 494, 549, 384]
Swaps: 0 Comparisons: 0
------------------------------------------------------------
------------------------------------------------------------
Pivot: 384 Low: 7 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 329, 439, 494, 549, 384]

Swapping pivot and high: 439 with 384

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]
Swaps: 1 Comparisons: 3
------------------------------------------------------------
```

```
------------------------------------------------------------------
Pivot: 494 Low: 9 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]

Swapping pivot and high: 549 with 494

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
Swaps: 0 Comparisons: 0
------------------------------------------------------------------

After Sorting: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
------------------------------------------------------------------

Total swaps: 4
Total comparisons: 23
```

Average Case: The average case run time of quick sort is O(n logn) . This case happens when we dont exactly get evenly balanced partitions. We might get at worst a 3-to-1 split on either side of pivot element.

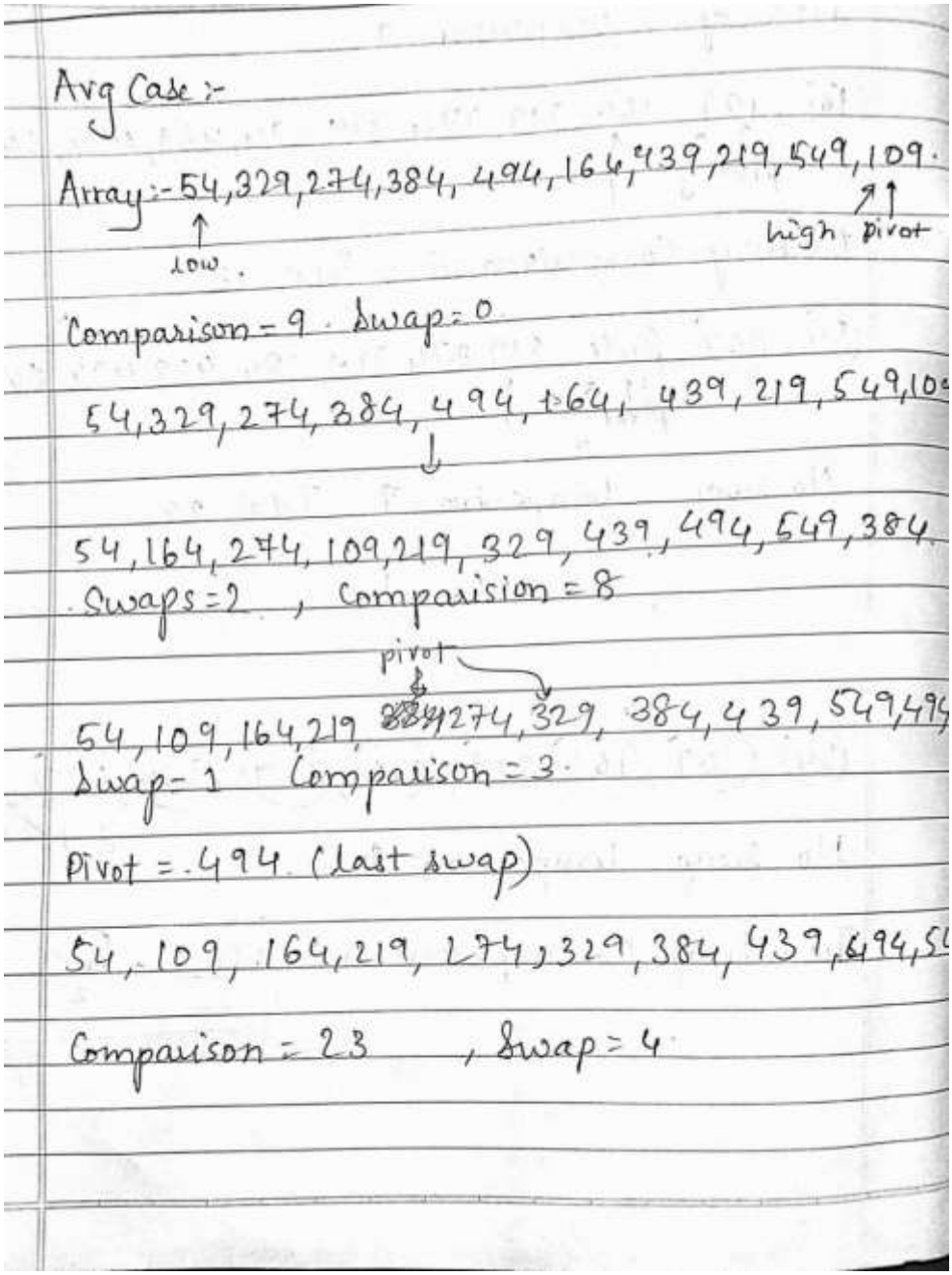


Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



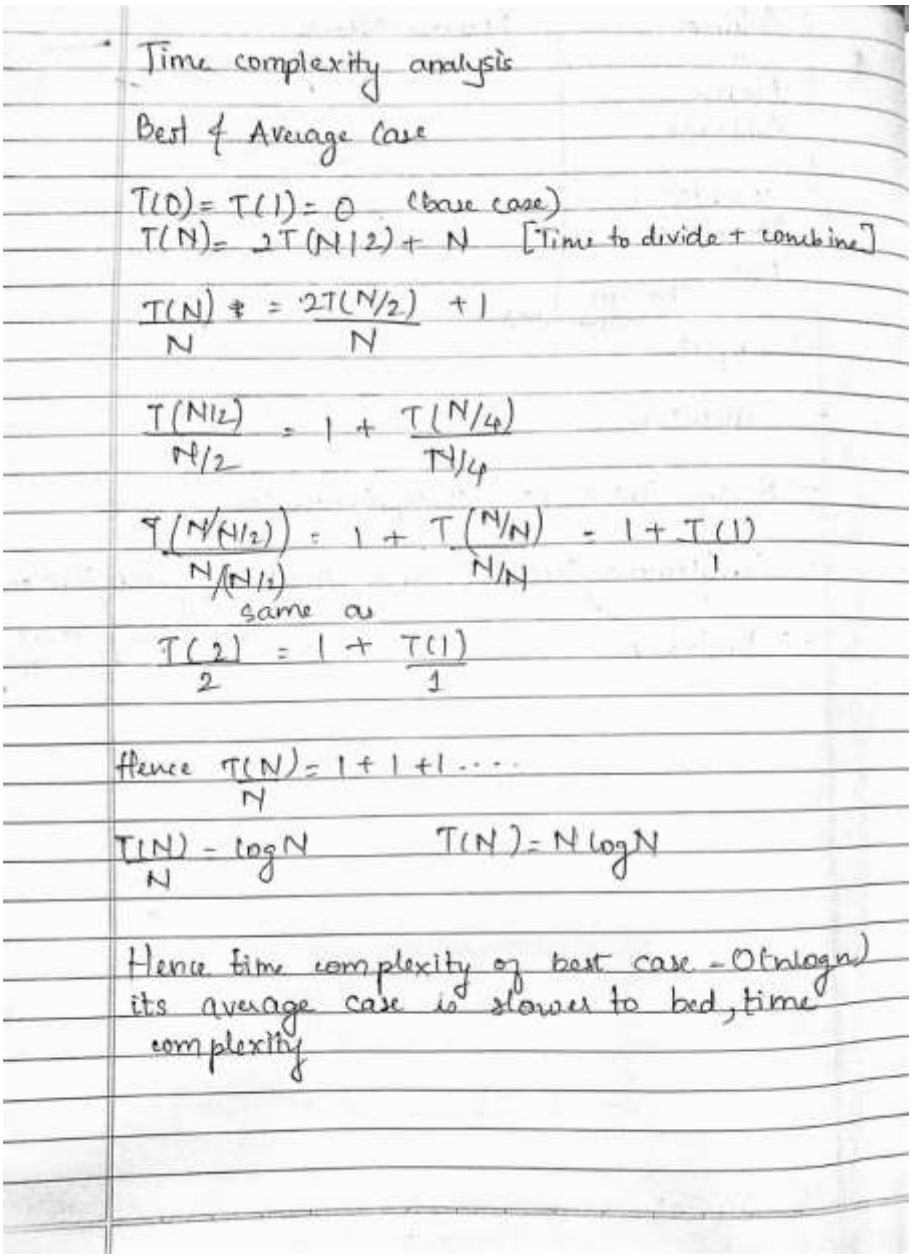
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Best case and Average case Time complexity analysis



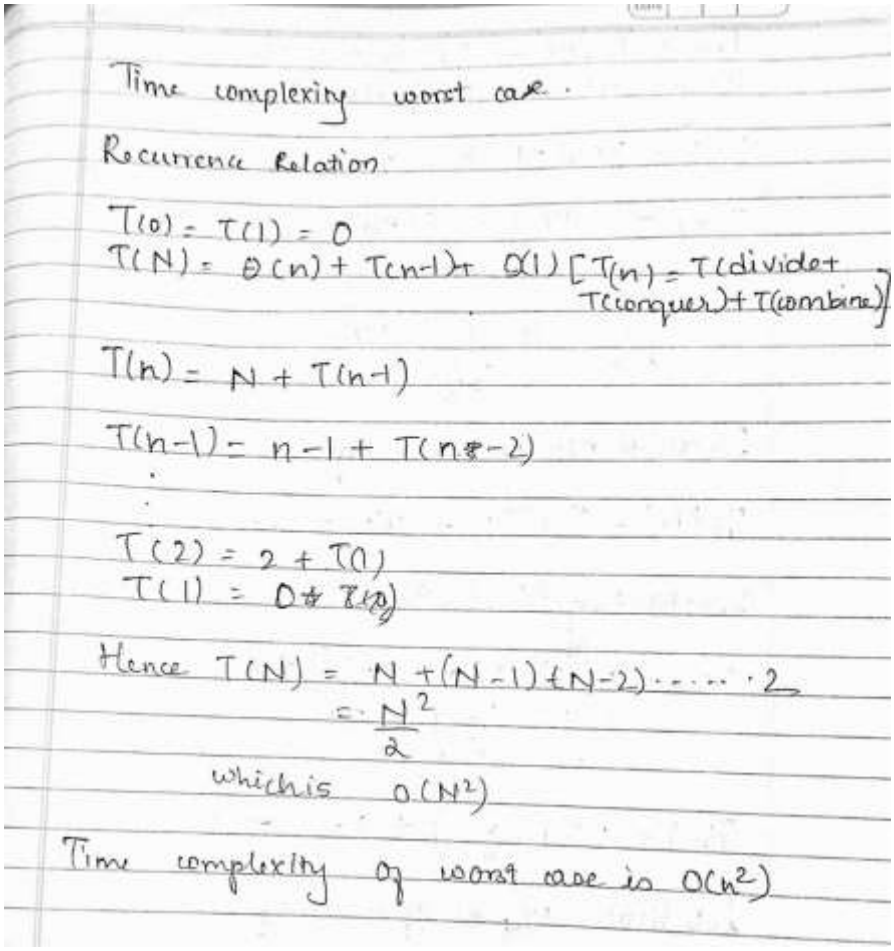
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Worst Case Time Complexity Analysis:



RESULT: Concepts gained from programming of problem:

- I have learnt about divide and conquer sorting algorithm like quicksort and mergesort
- Since the time complexity of following algo's are less than the O(n) which is O(nlogn) which makes them faster in case of large numbers of elements to sort as nlogn function has relatively constant slope at large values of N
- In quicksort the time complexity of any worst case is O(n2) while in mergesort its O(nlogn) for all cases
- I learnt how to use proper looping in order to avoid the ArrayOutOfBoundsException

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT (MergeSort)	
CODE:	import java.util.ArrayList; import java.util.Arrays; import java.util.Collections; import java.util.Scanner; public class mergeSortClass { *// Merge two subarrays L and M into arr* int shift = 0; void merge(int arr[], int p, int q, int r) { *// Create temp array to store the merged subarrays* int n1 = q - p + 1; int n2 = r - q; shift = 0; int L[] = new int[n1]; int M[] = new int[n2]; for (int i = 0; i < n1; i++) L[i] = arr[p + i]; for (int j = 0; j < n2; j++) M[j] = arr[q + 1 + j]; *// Maintain current index of sub-arrays and main array* int i, j, k; i = 0; j = 0; k = p; *// Until we reach either end of either L or M, pick larger among* *// elements L and M and place them in the correct position at A[p..r]* while (i < n1 && j < n2) { if (L[i] <= M[j]) { arr[k] = L[i];

```
            i++;
         } else {
            arr[k] = M[j];
            j++;
            shift++;
         }
         k++;
      }

      // When we run out of elements in either L or M,
      // pick up the remaining elements and put in A[p..r]
      while (i < n1) {
         arr[k] = L[i];
         i++;
         k++;
      }

      while (j < n2) {
         arr[k] = M[j];
         j++;
         k++;
      }
   }

   // Divide the array into two subarrays, sort them and
merge them
   void mergeSort(int arr[], int l, int r) {
      if (l < r) {
         System.out.print("\n----------------------------------
-----------------");
         // m is the point where the array is divided into two
subarrays
         int m = (l + r) / 2;
         System.out.print("\nThe midposition is: " + m);

         System.out.print("\nLeft half: " +
Arrays.toString(Arrays.copyOfRange(arr, l, m)));
         mergeSort(arr, l, m);
```

```
            System.out.print("\nRight half: " +
printArray(Arrays.copyOfRange(arr, m + 1, arr.length)));
            mergeSort(arr, m + 1, r);
            System.out.print("\nTotal Shift: " + shift);
            merge(arr, l, m, r);
            System.out.print("\nMerged: " + printArray(arr));
            System.out.print("\n----------------------------------
----------------");
        }
    }

    /* A utility function to print array of size n */
    public String printArray(int arr[]) {
        return Arrays.toString(arr);
    }

    // Driver program
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.print("\n                          MERGESORT");
        System.out.print("\n----------------------------------
--------------------------\n");
        System.out.print("\nEnter the roll no: ");
        int rollNo = input.nextInt();

        int array[];
        ArrayList<Integer> list = new ArrayList<Integer>();

        // Case input
        System.out.print("\n1.Random Case\n2.Worst
Case\n3.Manual Case  :");
        int choice = input.nextInt();
        if (choice == 2) {
            // Worst Case
            for (int i = 9; i >= 0; i--)
                list.add(rollNo + (rollNo + 1) * i);

        } else if (choice == 1) {
            // Random Case
```

```
            for (int i = 0; i < 10; i++)
                list.add(rollNo + (rollNo + 1) * i);
            Collections.shuffle(list);
        } else {
            // Manual Case
            System.out.print("\nEnter the elements: ");
            for (int i = 0; i < 10; i++)
                list.add(input.nextInt());
        }
        array = new int[10];
        for (int i = 0; i < 10; i++)
            array[i] = list.get(i);
        mergeSortClass ob = new mergeSortClass();
        ob.mergeSort(array, 0, array.length - 1);
        System.out.println("\nSorted array:" +
ob.printArray(array));
        input.close();
    }
}
```

Pseudocode for MergeSort

- Declare left and right var which will mark the extreme indices of the array
- Left will be assigned to 0 and right will be assigned to n-1
- Find mid = (left+right)/2
- Call mergeSort on (left,mid) and (mid+1,rear)
- Above will continue till left<right
- Then we will call merge on the 2 subproblems

Merge sort Algorithm

```
MergeSort(arr, left, right):
    if left > right
        return
    mid = (left+right)/2
    mergeSort(arr, left, mid)
    mergeSort(arr, mid+1, right)
    merge(arr, left, mid, right)
end
```

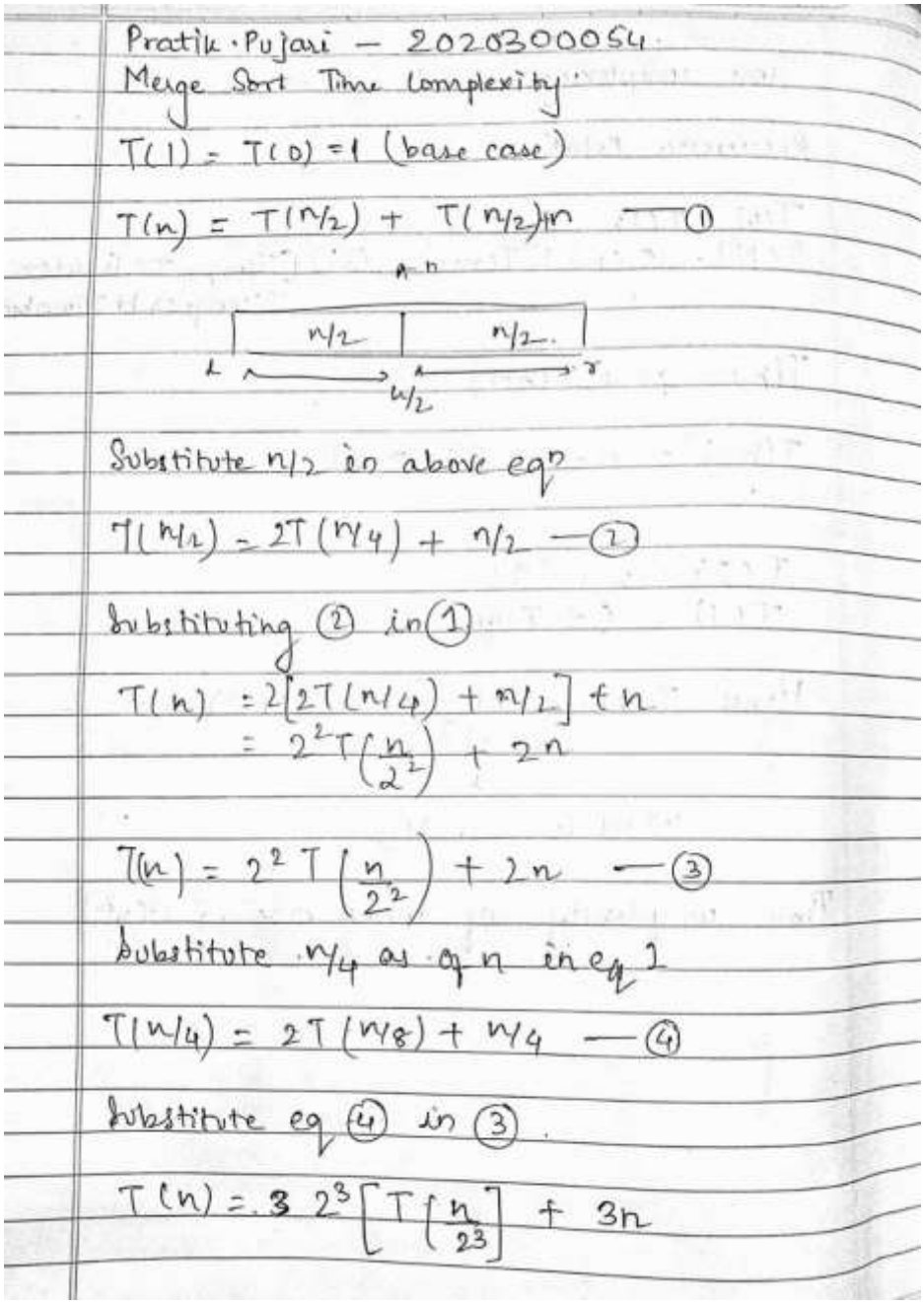
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

OUTPUT:

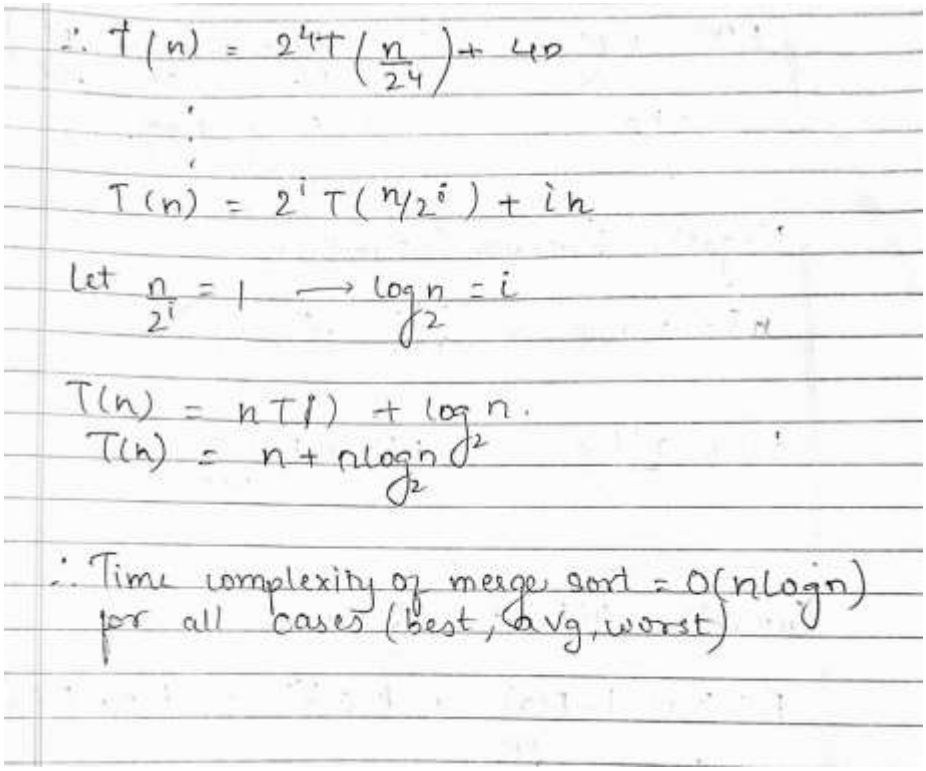


Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



OUTPUT:

Random Case:

```
                                MERGESORT
------------------------------------------------------------------------

Enter the roll no: 54

1.Random Case
2.Worst Case
3.Manual Case  :1

--------------------------------------------------------
The midposition is: 4
Left half: [54, 109, 494, 384]
--------------------------------------------------------
The midposition is: 2
Left half: [54, 109]
--------------------------------------------------------
The midposition is: 1
Left half: [54]
--------------------------------------------------------
```

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
-----------------------------------------------------
The midposition is: 0
Left half: []
Right half: [109, 494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
Total Shift: 0
Merged: [54, 109, 494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
Right half: [494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
Total Shift: 0
Merged: [54, 109, 494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
Right half: [384, 164, 549, 439, 219, 274, 329]
-----------------------------------------------------
The midposition is: 3
Left half: []
Right half: [164, 549, 439, 219, 274, 329]
=====================================================
Total Shift: 0
Merged: [54, 109, 494, 164, 384, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
=====================================================
Total Shift: 1
Merged: [54, 109, 164, 384, 494, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
Right half: [549, 439, 219, 274, 329]
-----------------------------------------------------
The midposition is: 7
Left half: [549, 439]
-----------------------------------------------------
The midposition is: 6
Left half: [549]
-----------------------------------------------------
The midposition is: 5
Left half: []
Right half: [439, 219, 274, 329]
=====================================================
Total Shift: 2
Merged: [54, 109, 164, 384, 494, 439, 549, 219, 274, 329]
=====================================================
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
------------------------------------------------------
Right half: [219, 274, 329]
======================================================
Total Shift: 1
Merged: [54, 109, 164, 384, 494, 219, 439, 549, 274, 329]
======================================================
------------------------------------------------------
Right half: [274, 329]
------------------------------------------------------
The midposition is: 8
Left half: []
Right half: [329]
======================================================
Total Shift: 1
Merged: [54, 109, 164, 384, 494, 219, 439, 549, 274, 329]
======================================================
------------------------------------------------------
======================================================
Total Shift: 0
Merged: [54, 109, 164, 384, 494, 219, 274, 329, 439, 549]
======================================================
------------------------------------------------------
======================================================
Total Shift: 2
Merged: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
======================================================
------------------------------------------------------
Sorted array:[54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	3		

AIM:	To implement the Bubble sort algorithm using Recurrence Relations concepts.
THEORY:	**What is Recursion?** The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc. **What is base condition in recursion?** In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems. int fact(int n) { if (n < = 1) // base case return 1; else return n*fact(n-1); } In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached. **What is the difference between direct and indirect recursion?** A function fun is called direct recursive if it calls the same

function fun. A function fun is called indirect recursive if it calls another function say fun_new and fun_new calls fun directly or indirectly. Difference between direct and indirect recursion has been illustrated in Table 1.

```
// An example of direct recursion
void directRecFun()
{
   // Some code....
   directRecFun();
   // Some code...
}

// An example of indirect recursion
void indirectRecFun1()
{
   // Some code...
   indirectRecFun2();
   // Some code...
}
void indirectRecFun2()
{
   // Some code...
   indirectRecFun1();
   // Some code...
}
```

Recurrence Relation

A recurrence is an equation or inequality that describes a function in terms of its values on smaller inputs. To solve a Recurrence Relation means to obtain a function defined on the natural numbers that satisfy the recurrence.

For Example, the Worst Case Running Time T(n) of the MERGE SORT Procedures is described by the recurrence.

$T(n) = \theta(1)$ if $n=1$

$2T\left(\frac{n}{2}\right) + \theta(n)$ if $n>1$

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

What is Bubble Sort?

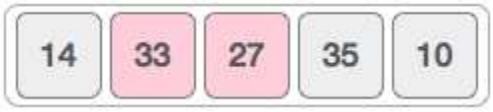
Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where **n** is the number of items.

How Bubble Sort Works?
We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this –

Next we compare 33 and 35. We find that both are in already sorted positions.

Then we move to the next two values, 35 and 10.

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

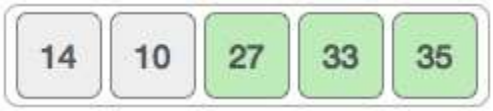
Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

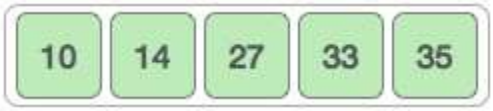
<table>
<tr>
<td></td>
<td>


We know then that 10 is smaller 35. Hence they are not sorted.


We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –


To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this –


Notice that after each iteration, at least one value moves at the end.


And when there's no swap required, bubble sorts learns that an array is completely sorted.


Now we should look into some practical aspects of bubble sort.
</td>
</tr>
<tr>
<td>PSEUDOCODE:</td>
<td>
begin BubbleSort(int arr[],int n):
for i=0 to i< (n-1)
if arr[i] is greater than arr[i+1]
swap arr[i] and arr[i+1]

if n-1 is greater than 1 return BubbleSort(arr,n-1)
</td>
</tr>
</table>

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	
CODE:	Bubble Sort code: import java.util.Arrays; public class BubbleSort { int counter = 0; int swapNum = 0; static int maxPos = 0; String swaps = "Swaps: |"; *// swap function* public static void swap(int[] arr, int i, int j) { int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp; } *// bubble sort function* public void bubbleSortIteration(int arr[]) { int n = arr.length; System.out.printf("Iteration\tSwap\t\tArray\n"); for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - i - 1; j++) { ++counter; if (arr[j] > arr[j + 1]) { *// swap arr[j+1] and arr[j]* System.out.printf("%d\t\t%d<- >%d\t\t%s\n", counter, arr[j], arr[j + 1], printArr(arr)); *// swappging the elements* int temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = temp; } else { System.out.printf("%d\t\t----\t\t%s\n", counter, printArr(arr));

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            }
         }
      }
   }

   public void bubbleSortRec(int[] arr, int n) {

      for (int i = 0; i < n - 1; i++) {
         if (arr[i] > arr[i + 1]) {
            ++swapNum;
            // swap arr[i+1] and arr[i]
            swaps += arr[i] + "<->" + arr[i + 1] + " | ";
            // swappging the elements
            swap(arr, i, i + 1);
         }
      }

      if (n - 1 > 1) {
         System.out.print("\n\n" + swaps);
         System.out.print(
               "\nUnsorted + Sorted => " + printArr(arr, 0,
n - 1) + " " + printArr(arr, n - 1, arr.length)
                     + "\n");
         swaps = "Swaps: |";
         bubbleSortRec(arr, n - 1);
      }
   }

   public int getSwapCount() {
      return swapNum;
   }

   public int[] bubbleSort(int i, int[] arr) {
      if (arr[i] > arr[i + 1] && (i + 1) < arr.length - 1) {
         swap(arr, i, i + 1);
         maxPos++;
      }
      if (i < arr.length - 1) {
```

```
            System.out.print("\n----------------------------------
------------------");
            System.out.println("\nMax Places " + arr[i] + " is
shifted " + maxPos + " times");
            System.out.println("\nArray: " + printArr(arr));
            maxPos = 0;
            bubbleSort(i + 1, arr);
        }
        return arr;
    }

    public String printArr(int arr[]) {
        // printing the array
        if (arr.length > 10)
            return " ... ";
        else
            return Arrays.toString(arr);
    }

    public String printArr(int[] arr, int start, int end) {
        // printing the array
        if (arr.length > 10)
            return " ... ";
        else
            return Arrays.toString(Arrays.copyOfRange(arr,
start, end));
    }
    // They need the array size case

}


Driver Code:
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

public class Driver {
```

```
   public static void main(String[] args) throws Exception {

      // Arraylist of Integers
      ArrayList<Integer> list = new ArrayList<Integer>();
      // array for sorting
      int[] array;
      // User input
      Scanner input = new Scanner(System.in);
      System.out.print("\n1.Random Array\n2.Roll.no
Input    : ");
      int choice = input.nextInt();
      if (choice == 1) {
         System.out.print("\nEnter the size of the array : ");
         int size = input.nextInt();
         for (int i = 0; i < size; i++) {
            list.add((int) (i + (Math.random() * 100)));
         }
         Collections.sort(list);
      } else {
         System.out.print("\nEnter the roll no: ");
         int roll = input.nextInt();
         for (int i = 0; i < 10; i++) {
            list.add(roll + (roll + 1) * i);
         }
      }
      System.out.print("\n1.Best Case\n2.Worst
Case\n3.Average Case\n4.Manual Choice\nEnter your
choice: ");
      int newChoice = input.nextInt();
      int listSize = list.size();
      array = new int[listSize];
      switch (newChoice) {
         case 1:
            for (int i = 0; i < listSize; i++) {
               array[i] = list.get(i);
            }
            break;
         case 2:
            for (int i = listSize - 1; i >= 0; i--) {
```

```
                array[(listSize - 1) - i] = list.get(i);
            }
            break;
        case 3:
            Collections.shuffle(list);
            for (int i = 0; i < listSize; i++) {
                array[i] = list.get(i);
            }
            break;
        case 4:
            if (choice == 1) {
                System.out.print("\nAre u sure u want to
enter the array manually?(y/n): ");
                String choice1 = input.next().toLowerCase();
                if (choice1.equals("y")) {
                    System.out.print("Enter the size of the
array: ");
                    int size = input.nextInt();
                    list.clear();
                    System.out.print("Enter the elements of
the array(with space): ");
                    for (int i = 0; i < size; i++) {
                        list.add(input.nextInt());
                    }
                }
            }
            break;
        default:
            System.out.println("Invalid choice");
            break;
    }
    int arrLen = array.length;
    // Array segregation
    switch (arrLen) {
        case 0:
            input.close();
            throw new Exception("Array is empty");
        case 1:
            input.close();
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            throw new Exception("Array has only one
element");
         default:
            System.out.print("\nArray: " +
Arrays.toString(array));
            break;
      }
      System.out
            .print("\n**********
**********\n");
      System.out.print("\nBubble Sort\n");
      // Bubble Sort
      System.out.print("\n1.Iteration Buuble
Sort\n2.Recursive Buuble Sort\n3.Bubble Sort (without for
loops)   : ");
      int choice2 = input.nextInt();
      // Create bubble sort object
      BubbleSort bs = new BubbleSort();
      // Call bubble sort function
      System.out.print("\nBefore sorting: " +
Arrays.toString(array));
      System.out.print("\n----------
----------\n");
      if (choice2 == 1) {
         bs.bubbleSortIteration(array);
      } else {
         bs.bubbleSortRec(array, listSize);
         System.out.print("\nTotal Swaps: " +
bs.getSwapCount());
      }

      System.out.print("\n----------
----------\n");
      System.out.print("\nFinal Array: " +
Arrays.toString(array) + " \n");
      // input closing method
      input.close();
   }
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

OUTPUT:

Best Case:

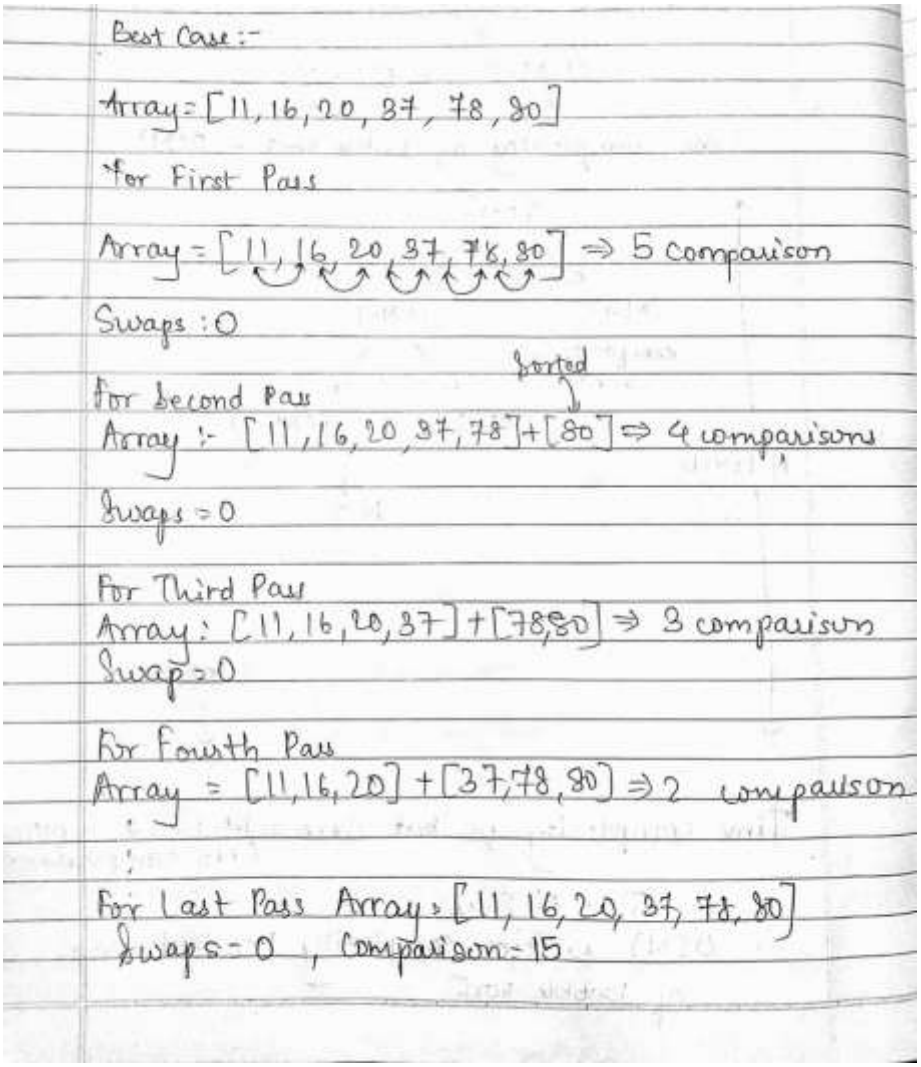
```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 6

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 1

Array: [11, 16, 20, 37, 78, 90]
**********************************************

Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 2

Before sorting: [11, 16, 20, 37, 78, 90]
----------------------------------------------


Swaps: |
Unsorted + Sorted => [11, 16, 20, 37, 78] [90]


Swaps: |
Unsorted + Sorted => [11, 16, 20, 37] [78, 90]


Swaps: |
Unsorted + Sorted => [11, 16, 20] [37, 78, 90]


Swaps: |
Unsorted + Sorted => [11, 16] [20, 37, 78, 90]

Total Swaps: 0
----------------------------------------------

Final Array: [11, 16, 20, 37, 78, 90]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



Average Case:

```
1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 3

Array: [99, 55, 5, 36, 6, 97, 98]
**********************************************
```

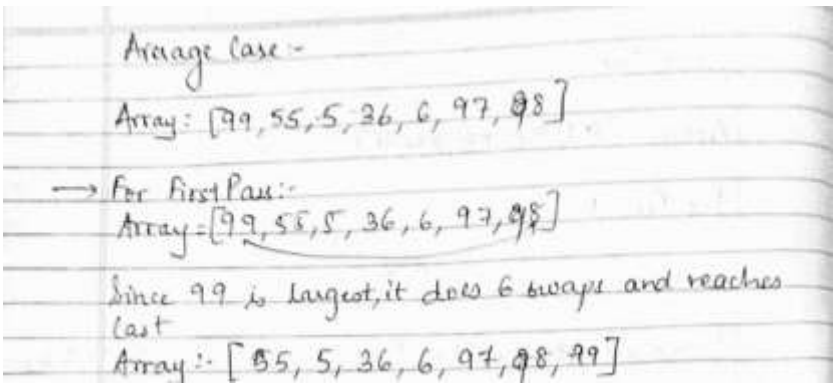
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

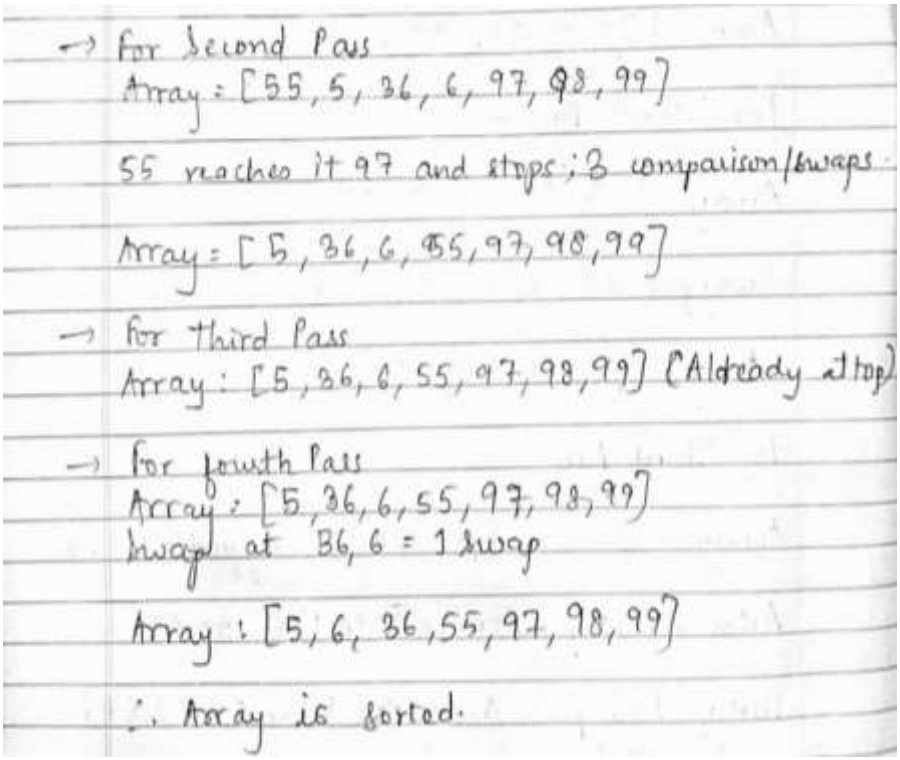
```
Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 3

Before sorting: [99, 55, 5, 36, 6, 97, 98]
-------------------------------------------


Swaps: |99<->55 | 99<->5 | 99<->36 | 99<->6 | 99<->97 | 99<->98 |
Unsorted + Sorted => [55, 5, 36, 6, 97, 98] [99]


Swaps: |55<->5 | 55<->36 | 55<->6 |
Unsorted + Sorted => [5, 36, 6, 55, 97] [98, 99]


Swaps: |36<->6 |
Unsorted + Sorted => [5, 6, 36, 55] [97, 98, 99]


Swaps: |
Unsorted + Sorted => [5, 6, 36] [55, 97, 98, 99]


Swaps: |
Unsorted + Sorted => [5, 6] [36, 55, 97, 98, 99]

Total Swaps: 10
-------------------------------------------

Final Array: [5, 6, 36, 55, 97, 98, 99]
```



Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



Worst Case:

```
Enter the size of the array : 5

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 2

Array: [87, 82, 58, 24, 1]
************************************************
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 2

Before sorting: [87, 82, 58, 24, 1]
-------------------------------------------


Swaps: |87<->82 | 87<->58 | 87<->24 | 87<->1 |
Unsorted + Sorted => [82, 58, 24, 1] [87]


Swaps: |82<->58 | 82<->24 | 82<->1 |
Unsorted + Sorted => [58, 24, 1] [82, 87]


Swaps: |58<->24 | 58<->1 |
Unsorted + Sorted => [24, 1] [58, 82, 87]

Total Swaps: 10
-------------------------------------------

Final Array: [1, 24, 58, 82, 87]
```

Time Complexity of Bubble Sort:

Worst Case Time Complexity

Θ(N^2) is the Worst Case Time Complexity of Bubble Sort. This is the case when the array is reversely sort
The number of swaps of two elements is equal to the number of comparisons in this case as every element is out of place.

$T(N)=C(N)=S(N)=N*(N-1)2T(N)=C(N)=S(N)=N*(N-1)2$

Therefore, in the worst case:

- Number of Comparisons: O(N^2) time
- Number of swaps: O(N^2) time
-

Best Case Time Complexity

Θ(N) is the Best Case Time Complexity of Bubble Sort.
This case occurs when the given array is already sorted.
T(N)=C(N)=NT(N)=C(N)=N
S(N)=0S(N)=0

Therefore, in the best case:

- Number of Comparisons: N = O(N) time
- Number of swaps: 0 = O(1) time

Average Case Time Complexity

Θ(N^2) is the Average Case Time Complexity of Bubble Sort.
The number of comparisons is constant in Bubble Sort so in average case, there is O(N^2) comparisons. This is because irrespective of the arrangement of elements, the number of comparisons C(N) is same.
For the number of swaps, consider the following points:

- If an element is in index I1 but it should be in index I2, then it will take a minimum of I2-I1 swaps to bring the element to the correct position.
- An element E will be at a distance of I3 from its position in sorted array
- The sum of maximum difference in position across all elements will be:

(N-1) + (N-3) + (N-5) ... + 0 + ... + (N-3) + (N-1)
= N x N - 2 x (1 + 3 + 5 + ... + N/2)
= N^2 - 2 x N^2 / 4
= N^2 - N^2 / 2
= N^2 / 2
Therefore, in average, the number of swaps = O(N^2).
Therefore, in the average case time complexity of Bubble sort:

- Number of Comparisons: O(N^2) time
- Number of swaps: O(N^2) time

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

TIME COMPLEXITY:

h=5

Pratik Pujari 2020300054

```
public void BubbleSort(int arr[], int n){
{
  for(int i=0; i<(n-1); i++)
  {
    if (arr[i] > arr[i+1]){

      int temp = arr[i];
      arr[i] = arr[i+1];
      arr[i+1] = temp;
    }
  }
  if (n-1) > 1){

  BubbleSort(arr, n-1)
  }
}
```

Time complexity of Recursive Bubble Sort

$T(N) = T(N-1) + (N-1)$ → For loop.
↑ recursive return

$T(N-1) = T(N-2) + (N-2)$

$T(1) = 1$

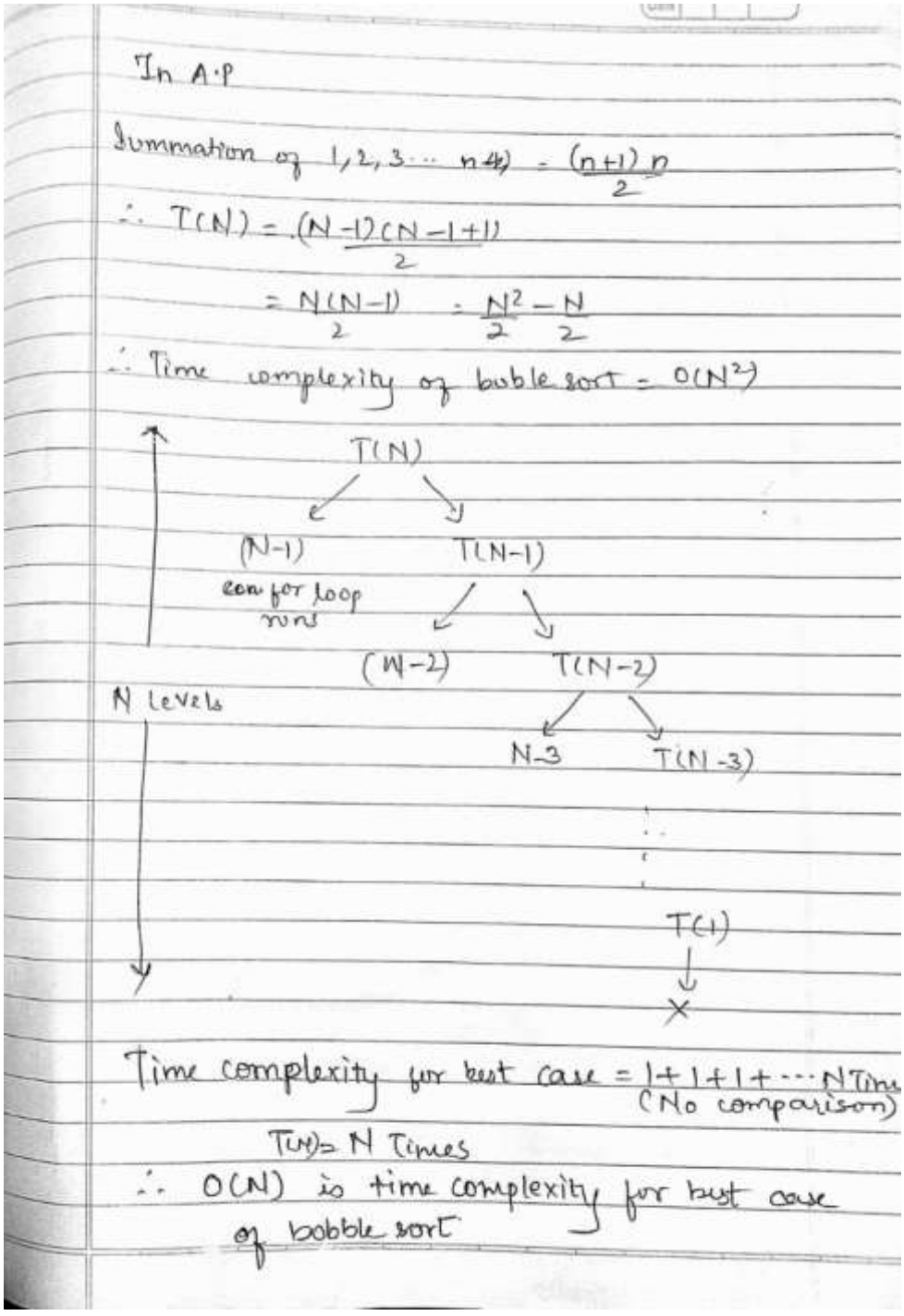
$\therefore T(N) = (N-1) + (N-2) \cdots\cdots 1$

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



RESULT: Things learnt during procedural programming during solving of the problem

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

- Learnt how to implement bubble sort using iteration and recursion
- Learnt how to use exception in order to show that the array is of invalid length
- Learnt different time complexity cases of bubble sort and how it can be used.
- Learnt bubble sort required more time and space as it go through the array N^2 times

Extra Outputs:

```
Swaps: |67<->41 | 67<->16 | 67<->26 | 83<->34 | 83<->27 | 83<->71 | 83<->49 | 83<->15 | 83<->82 |
 83<->60 | 83<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |41<->16 | 41<->26 | 67<->34 | 67<->27 | 71<->49 | 71<->15 | 82<->60 | 82<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |41<->34 | 41<->27 | 67<->49 | 67<->15 | 71<->60 | 71<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |34<->27 | 49<->15 | 67<->60 | 67<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |41<->15 | 60<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |34<->15 | 49<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |27<->15 | 41<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |26<->15 | 34<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |16<->15 | 27<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |26<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |16<->15 |
Unsorted + Sorted =>  ...   ...

Total Swaps: 53
----------------------------------------

Final Array: [15, 15, 16, 26, 27, 34, 41, 49, 60, 67, 71, 82, 83, 86]
```

(If the array is too big it shows the '….')

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

(When the array size is 0/1 it shows error)

```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 0

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 1
Exception in thread "main" java.lang.Exception: Array is empty
        at Driver.main(Driver.java:78)
```

(User gets a chance to renter array)

```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 5

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 4

Are u sure u want to enter the array manually?(y/n): y
Enter the size of the array: 5
Enter the elements of the array(with space): 1 3 6 2 7

Array: [1, 3, 6, 2, 7]
************************************************
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

(Iterative Bubble sort)

```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 6

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 1

Array: [20, 75, 91, 93, 96, 97]
**********************************************

Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 1

Before sorting: [20, 75, 91, 93, 96, 97]
----------------------------------------------
Iteration       Swap            Array
1               ----            [20, 75, 91, 93, 96, 97]
2               ----            [20, 75, 91, 93, 96, 97]
3               ----            [20, 75, 91, 93, 96, 97]
4               ----            [20, 75, 91, 93, 96, 97]
5               ----            [20, 75, 91, 93, 96, 97]
6               ----            [20, 75, 91, 93, 96, 97]
7               ----            [20, 75, 91, 93, 96, 97]
8               ----            [20, 75, 91, 93, 96, 97]
9               ----            [20, 75, 91, 93, 96, 97]
10              ----            [20, 75, 91, 93, 96, 97]
11              ----            [20, 75, 91, 93, 96, 97]
12              ----            [20, 75, 91, 93, 96, 97]
13              ----            [20, 75, 91, 93, 96, 97]
14              ----            [20, 75, 91, 93, 96, 97]
15              ----            [20, 75, 91, 93, 96, 97]

----------------------------------------------

Final Array: [20, 75, 91, 93, 96, 97]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

(Roll no input)

```
Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 2

Before sorting: [439, 109, 164, 329, 384, 274, 219, 549, 54, 494]
-------------------------------------------


Swaps: |439<->109 | 439<->164 | 439<->329 | 439<->384 | 439<->274 | 439<->219 | 549<->54 | 549
<->494 |
Unsorted + Sorted => [109, 164, 329, 384, 274, 219, 439, 54, 494] [549]


Swaps: |384<->274 | 384<->219 | 439<->54 |
Unsorted + Sorted => [109, 164, 329, 274, 219, 384, 54, 439] [494, 549]


Swaps: |329<->274 | 329<->219 | 384<->54 |
Unsorted + Sorted => [109, 164, 274, 219, 329, 54, 384] [439, 494, 549]


Swaps: |274<->219 | 329<->54 |
Unsorted + Sorted => [109, 164, 219, 274, 54, 329] [384, 439, 494, 549]


Swaps: |274<->54 |
Unsorted + Sorted => [109, 164, 219, 54, 274] [329, 384, 439, 494, 549]


Swaps: |219<->54 |
Unsorted + Sorted => [109, 164, 54, 219] [274, 329, 384, 439, 494, 549]


Swaps: |164<->54 |
Unsorted + Sorted => [109, 54, 164] [219, 274, 329, 384, 439, 494, 549]


Swaps: |109<->54 |
Unsorted + Sorted => [54, 109] [164, 219, 274, 329, 384, 439, 494, 549]

Total Swaps: 20
-------------------------------------------

Final Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	4		

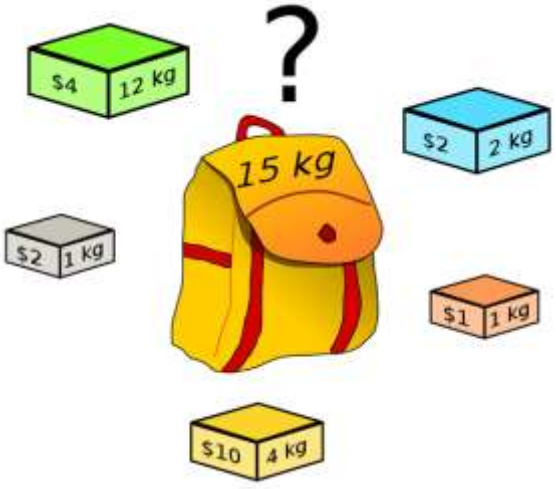
AIM:	To implement problems of Dynamic Programming
THEORY:	**What is Dynamic Programming?** Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems. **Characteristics of Dynamic Programming** Before moving on to understand different methods of solving a DP problem, let's first take a look at what are the characteristics of a problem that tells us that we can apply DP to solve it. **Top-down with Memoization** In this approach, we try to solve the bigger problem by recursively finding the solution to smaller sub-problems. Whenever we solve a sub-problem, we cache its result so that we don't end up solving it repeatedly if it's called multiple times. Instead, we can just return the saved result. This technique of storing the results of already solved subproblems is called Memoization. **Bottom-up with Tabulation** Tabulation is the opposite of the top-down approach and avoids recursion. In this approach, we solve the problem "bottom-up" (i.e. by solving all the related sub-problems first). This is typically done by filling up an n-dimensional

table. Based on the results in the table, the solution to the top/original problem is then computed.
Tabulation is the opposite of Memoization, as in Memoization we solve the problem and maintain a map of already solved sub-problems. In other words, in memoization, we do it top-down in the sense that we solve the top problem first (which typically recurses down to solve the sub-problems).

What is the Knapsack Problem?

The Knapsack Problem is a famous *Dynamic Programming* Problem that falls in the optimization category.
It derives its name from a scenario where, given a set of items with specific weights and assigned values, the goal is to maximize the value in a knapsack while remaining within the weight constraint.
Each item can only be selected once, as we don't have multiple quantities of any item.

Problem:
Given a Knapsack of a maximum capacity of W and N items each with its own value and weight, throw in items inside the Knapsack such that the final contents has the maximum value. Yikes !!



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Here's the general way the problem is explained – Consider a thief gets into a home to rob and he carries a knapsack. There are fixed number of items in the home – each with its own weight and value – Jewellery, with less weight and highest value vs tables, with less value but a lot heavy. To add fuel to the fire, the thief has an old knapsack which has limited capacity. Obviously, he can't split the table into half or jewellery into 3/4ths. He either takes it or leaves it

Example:
Knapsack Max weight: W = 10 (units)

Total items: N = 4

Values of items: v[] = {10, 40, 30, 50}

Weight of items: w[] = {5, 4, 6, 3}
A cursory look at the example data tells us that the max value that we could accommodate with the limit of max weight of 10 is 50 + 40 = 90 with a weight of 7.
Approach:
The way this is optimally solved is using dynamic programming – solving for smaller sets of knapsack problems and then expanding them for the bigger problem. Let's build an Item x Weight array called V (Value array):
V[N][W] = 4 rows * 10 columns
Each of the values in this matrix represent a smaller Knapsack problem.
Base case 1 : Let's take the case of 0th column. It just means that the knapsack has 0 capacity. What can you hold in them? Nothing. So, let's fill them up all with 0s.
Base case 2 : Let's take the case of 0 row. It just means that there are no items in the house. What do you do hold in your knapsack if there are no items. Nothing again !!! All zeroes.

Value Array - Item 0, Weight 0

	Weight 0	Weight 1	Weight 2	Weight 3	Weight 4	Weight 5	Weight 6	Weight 7	Weight 8	Weight 9	Weight 10
Item 0	0	0	0	0	0	0	0	0	0	0	0
Item 1	0										
Item 2	0										
Item 3	0										
Item 4	0										

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Solution:

1) Now, let's start filling in the array row-wise. What does row 1 and column 1 mean? That given the first item (row), can you accommodate it in the knapsack with capacity 1 (column). Nope. The weight of the first item is 5. So, let's fill in 0. In fact, we wouldn't be able to fill in anything until we reach the column 5 (weight 5).

2) Once we reach column 5 (which represents weight 5) on the first row, it means that we could accommodate item 1. Let's fill in 10 there (remember, this is a Value array):

Value Array - Item 1, Weight 5

	Weight 0	Weight 1	Weight 2	Weight 3	Weight 4	Weight 5	Weight 6	Weight 7	Weight 8	Weight 9	Weight 10
Item 0	0	0	0	0	0	0	0	0	0	0	0
Item 1	0	0	0	0	0	10					
Item 2	0										
Item 3	0										
Item 4	0										

3) Moving on, for weight 6 (column 6), can we accommodate anything else with the remaining weight of 1 (weight – weight of this item => 6 – 5). Hey, remember, we are on the first item. So, it is kind of intuitive that the rest of the row will just be the same value too since we are unable to add in any other item for that extra weight that we have got.

Value Array - Item 1, Weight 6

	Weight 0	Weight 1	Weight 2	Weight 3	Weight 4	Weight 5	Weight 6	Weight 7	Weight 8	Weight 9	Weight 10
Item 0	0	0	0	0	0	0	0	0	0	0	0
Item 1	0	0	0	0	0	10	10				
Item 2	0										
Item 3	0										
Item 4	0										

4) So, the next interesting thing happens when we reach the column 4 in third row. The current running weight is 4. We should check for the following cases.

1) Can we accommodate Item 2 – Yes, we can. Item 2's weight is 4.

2) Is the value for the current weight is higher without Item 2? – Check the previous row for the same weight. Nope. the previous row* has 0 in it, since we were not able able accommodate Item 1 in weight 4.

3) Can we accommodate two items in the same weight so that we could maximize the value? – Nope. The remaining weight after deducting the Item2's weight is 0.

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Value Array - Item 2, Weight 4

	Weight 0	Weight 1	Weight 2	Weight 3	Weight 4	Weight 5	Weight 6	Weight 7	Weight 8	Weight 9	Weight 10
Item 0	0	0	0	0	0	0	0	0	0	0	0
Item 1	0	0	0	0	0	10	10	10	10	10	10
Item 2	0	0	0	0	40						
Item 3	0										
Item 4	0										

Why previous row?

Simply because the previous row at weight 4 itself is a smaller knapsack solution which gives the max value that could be accumulated for that weight until that point (traversing through the items).

Exemplifying,

1) The value of the current item = 40
2) The weight of the current item = 4
3) The weight that is left over = 4 – 4 = 0
4) Check the row above (the Item above in case of Item 1 or the cumulative Max value in case of the rest of the rows). For the remaining weight 0, are we able to accommodate Item 1? Simply put, is there any value at all in the row above for the given weight?

The calculation goes like so :

1) Take the max value for the same weight without this item:
previous row, same weight = 0
=> V[item-1][weight]
2) Take the value of the current item + value that we could accommodate with the remaining weight:
Value of current item
+ value in previous row with weight 4 (total weight until now (4) - weight of the current item (4))
=> val[item-1] + V[item-1][weight-wt[item-1]]
Max among the two is 40 (0 and 40).
3) The next and the most important event happens at column 9 and row 2. Meaning we have a weight of 9 and we have two items. Looking at the example data we could accommodate the first two items. Here, we consider few things:

1. The value of the current item = 40
2. The weight of the current item = 4
3. The weight that is left over = 9 - 4 = 5

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

<table>
<tr>
<td></td>
<td>
4. Check the row above. At the remaining weight 5, are we able to accommodate Item 1.
Value Array - Item 2, Weight 9 (accomodates 1 and 2)
<table>
<tr><th></th><th>Weight 0</th><th>Weight 1</th><th>Weight 2</th><th>Weight 3</th><th>Weight 4</th><th>Weight 5</th><th>Weight 6</th><th>Weight 7</th><th>Weight 8</th><th>Weight 9</th><th>Weight 10</th></tr>
<tr><td>Item 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Item 1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td></tr>
<tr><td>Item 2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>40</td><td>40</td><td>40</td><td>40</td><td>40</td><td>50</td><td></td></tr>
<tr><td>Item 3</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Item 4</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
So, the calculation is :
1) Take the max value for the same weight without this item:
previous row, same weight = 10
2) Take the value of the current item + value that we could accumulate with the remaining weight:
Value of current item (40)
+ value in previous row with weight 5 (total weight until now (9) - weight of the current item (4))= 10
10 vs 50 = 50.
At the end of solving all these smaller problems, we just need to return the value at V[N][W] – Item 4 at Weight 10:
Value Array - Final set
<table>
<tr><th></th><th>Weight 0</th><th>Weight 1</th><th>Weight 2</th><th>Weight 3</th><th>Weight 4</th><th>Weight 5</th><th>Weight 6</th><th>Weight 7</th><th>Weight 8</th><th>Weight 9</th><th>Weight 10</th></tr>
<tr><td>Item 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Item 1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td></tr>
<tr><td>Item 2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>40</td><td>40</td><td>40</td><td>40</td><td>40</td><td>50</td><td>50</td></tr>
<tr><td>Item 3</td><td>0</td><td>0</td><td>0</td><td>0</td><td>40</td><td>40</td><td>40</td><td>40</td><td>40</td><td>50</td><td>70</td></tr>
<tr><td>Item 4</td><td>0</td><td>0</td><td>0</td><td>50</td><td>50</td><td>50</td><td>50</td><td>90</td><td>90</td><td>90</td><td>90</td></tr>
</table>
Complexity
Analysing the complexity of the solution is pretty straight-forward. We just have a loop for W within a loop of N => O (N*W)
</td>
</tr>
<tr>
<td>PSEUDOCODE:</td>
<td>
array m[0..n, 0..W];
for j from 0 to W do:
m[0, j] := 0
for i from 1 to n do:
m[i, 0] := 0

for i from 1 to n do:
for j from 0 to W do:
if w[i] > j then:
m[i, j] := m[i-1, j]
else:
m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])
</td>
</tr>
</table>

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	
CODE:	import java.util.Scanner; import java.io.BufferedReader; import java.util.ArrayList; public class KnapSack { public static void listtoArray(int array[], ArrayList<Integer> list) { for (int i = 0; i < list.size(); i++) { array[i] = list.get(i); } } public static void main(String[] args) throws Exception { Scanner input = new Scanner(System.in); *// Arraylist for storing temp values* ArrayList<Integer> _temp = new ArrayList<Integer>(); String numbers[]; int length = 0; int values[], weight[], W; System.out.print("\n\t\tKNAPSACK ALGORITHM"); BufferedReader br = new BufferedReader(new java.io.InputStreamReader(System.in)); *// Take values input* System.out.print("\nEnter the Values of the array(with space)\n-> "); String inputstr = br.readLine(); if (!inputstr.equals("")) { numbers = inputstr.split(" "); for (String number : numbers) _temp.add(Integer.parseInt(number)); length = _temp.size(); }

```
        values = new int[length];
        listtoArray(values, _temp);

        // weight input
        weight = new int[length];
        System.out.print("\n---- Weights of items ----\n ");
        for (int i = 0; i < length; i++) {
            System.out.print("\n-> Weight of item '" + _temp.get(i) + "' : ");
            weight[i] = input.nextInt();
        }

        // Maz weight
        System.out.print("\nEnter the Max Weight: ");
        W = input.nextInt();

        System.out.println("\nThe limit of max possible weight is " +
knapsack(values, weight, W));
        input.close();

    }

    public static int knapsack(int val[], int wt[], int W) {

        System.out.print("\nFormulating the problem \n");

        // Get the total number of items.
        // Could be wt.length or val.length. Doesn't matter

        int N = wt.length;

        // Create a matrix.
        // Items are in rows and weight at in columns +1 on each side

        int[][] values = new int[N + 1][W + 1];

        // What if the knapsack's capacity is 0 - Set
        // all columns at row 0 to be 0

        for (int col = 0; col <= W; col++) {
```

```
        values[0][col] = 0;

    }

    // What if there are no items at home.
    // Fill the first row with 0
    for (int row = 0; row <= N; row++) {
        values[row][0] = 0;
    }

    for (int item = 1; item <= N; item++) {

        // Let's fill the values row by row
        for (int weight = 1; weight <= W; weight++) {
            // Is the current items weight less
            // than or equal to running weight
            if (wt[item - 1] <= weight) {

                // Given a weight, check if the value of the current
                // item + value of the item that we could afford
                // with the remaining weight is greater than the value
                // without the current item itself

                values[item][weight] = Math.max(val[item - 1] + values[item -
1][weight - wt[item - 1]],
                        values[item - 1][weight]);
            }

            else {
                // If the current item's weight is more than the
                // running weight, just carry forward the value
                // without the current item

                values[item][weight] = values[item - 1][weight];
            }
        }

    }
```

Computer Engineering Department &
Information Technology Engineering Department

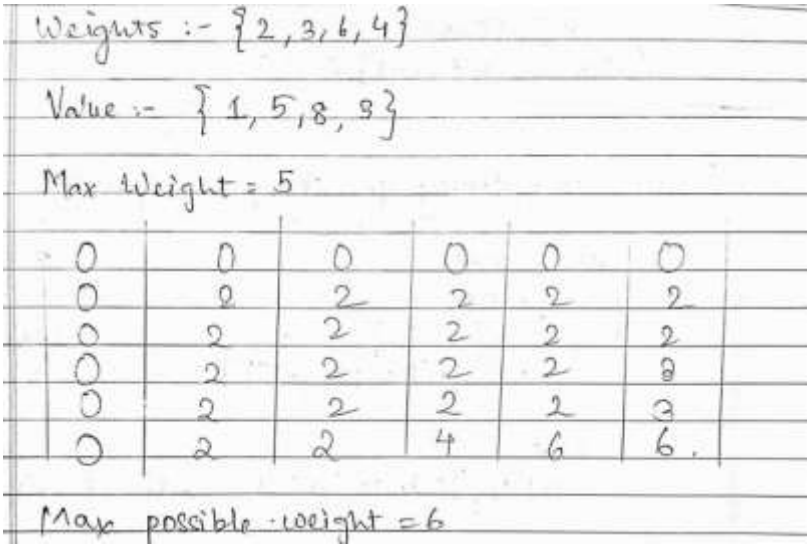
Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        // Printing the matrix
        for (int[] rows : values) {
            for (int col : rows) {
                System.out.format("%5d", col);
            }
            System.out.println();
        }
        return values[N][W];
    }

}
```

OUTPUT:

```
                KNAPSACK ALGORITHM
Enter the Values of the array(with space)
-> 2 5 8 1

---- Weights of items ----

-> Weight of item '2' : 3

-> Weight of item '5' : 6

-> Weight of item '8' : 8

-> Weight of item '1' : 1

Enter the Max Weight: 18

Formulating the problem
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
    0    0    0    2    2    2    5    5    5    7    7    7    7    7    7    7    7    7    7
    0    0    0    2    2    2    5    5    8    8    8   10   10   10   13   13   13   15   15
    0    0    0    2    2    2    5    5    8    8    8   10   10   10   13   13   13   15   15

The limit of max possible weight is 15
```



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Input:
Weights: 0
Max Wgt: 1

```
                KNAPSACK ALGORITHM
Enter the Values of the array(with space)
->

---- Weights of items ----

Enter the Max Weight: 1

Formulating the problem
    0    0

The limit of max possible weight is 0
```

No of weights: 1, Max Weight: 5
Weight: 10
Weight val1: 5

```
                    KNAPSACK ALGORITHM
Enter the Values of the array(with space)
-> 10

---- Weights of items ----

-> Weight of item '10' : 5

Enter the Max Weight: 5

Formulating the problem
    0    0    0    0    0    0
    0    0    0    0    0   10

The limit of max possible weight is 10
```

No of weights: 4 Max Weight: 10
Weights: 10 20 30 40
Values: 12 13 15 19

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
                KNAPSACK ALGORITHM
Enter the Values of the array(with space)
-> 10 20 30 40

---- Weights of items ----

-> Weight of item '10' : 12

-> Weight of item '20' : 13

-> Weight of item '30' : 15

-> Weight of item '40' : 19

Enter the Max Weight: 10

Formulating the problem
    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0

The limit of max possible weight is 0
```

No of weights: 4, Max Weight: 7
Weights: 80 60 40 20
Values: 2 4 6 8

```
                KNAPSACK ALGORITHM
Enter the Values of the array(with space)
-> 80 60 40 20

---- Weights of items ----

-> Weight of item '80' : 2

-> Weight of item '60' : 4

-> Weight of item '40' : 6

-> Weight of item '20' : 8

Enter the Max Weight: 7

Formulating the problem
    0    0    0    0    0    0    0    0
    0    0   80   80   80   80   80   80
    0    0   80   80   80   80  140  140
    0    0   80   80   80   80  140  140
    0    0   80   80   80   80  140  140

The limit of max possible weight is 140
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

No of weights: 5, max Weight: 15
Weights: 4 2 1 10 2
Values: 12 2 1 4 1

```
                KNAPSACK ALGORITHM
Enter the Values of the array(with space)
-> 4 2 1 10 2

---- Weights of items ----

-> Weight of item '4' : 12

-> Weight of item '2' : 2

-> Weight of item '1' : 1

-> Weight of item '10' : 4

-> Weight of item '2' : 1

Enter the Max Weight: 15

Formulating the problem
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   4   4   4   4
    0   0   2   2   2   2   2   2   2   2   2   2   4   4   6   6
    0   1   2   3   3   3   3   3   3   3   3   3   4   5   6   7
    0   1   2   3  10  11  12  13  13  13  13  13  13  13  13  13
    0   2   3   4  10  12  13  14  15  15  15  15  15  15  15  15

The limit of max possible weight is 15
```

No of weights: 10, Max weight: 165
Weight: 92 57 49 68 60 43 67 84 87 72
Values: 23 31 29 44 53 38 63 85 89 82

```
                KNAPSACK ALGORITHM
Enter the Values of the array(with space)
-> 92 57 49 68 60 43 67 84 87 72

---- Weights of items ----

-> Weight of item '92' : 23

-> Weight of item '57' : 31

-> Weight of item '49' : 29

-> Weight of item '68' : 44

-> Weight of item '60' : 53

-> Weight of item '43' : 38

-> Weight of item '67' : 63

-> Weight of item '84' : 85

-> Weight of item '87' : 89

-> Weight of item '72' : 82

Enter the Max Weight: 165
```

Output:

```
The limit of max possible weight is 309
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

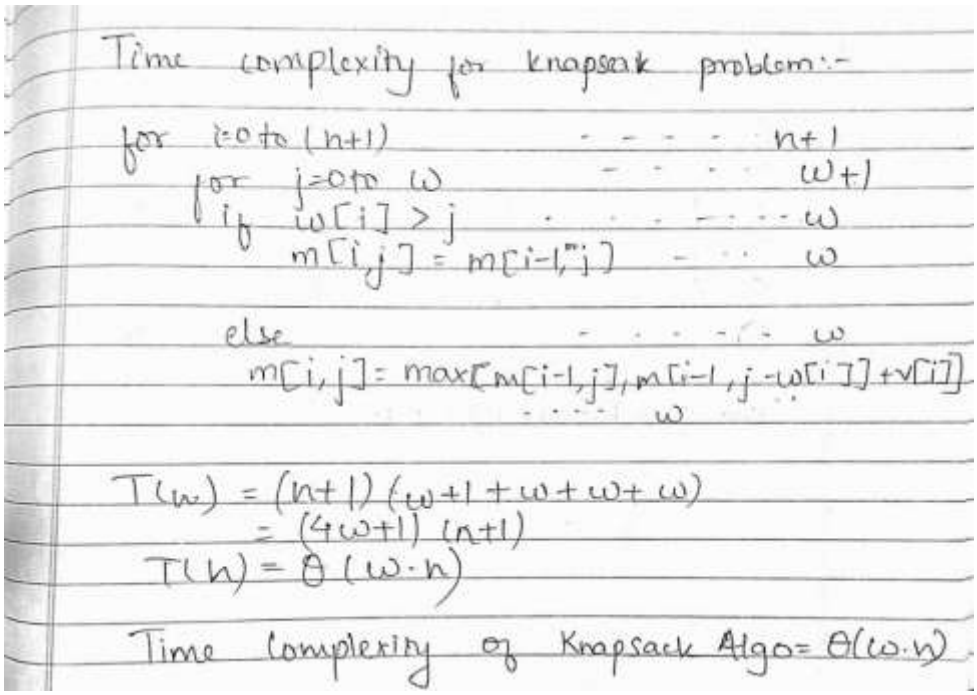
<table>
<tr>
<td></td>
<td>No of weight: 15, max Weight: 750
Weight:
135 139 149 150 156 163 173 184 192 201 210 214 221 229 240
Values: 70 73 77 80 82 87 90 94 98 106 110 113 115 118 120

The limit of max possible weight is 1458

No of weight: 24, Max Weight: 6404180
Weight: 825594 1677009 1676628 1523970 943972 97426 69666 1296457 1679693 1902996 1844992 1049289 1252836 1319836 953277 2067538 675367 853655 1826027 65731 901489 577243 466257 369261

Values: 382745 799601 909247 729069 467902 44328 34610 698150 823460 903959 853665 551830 610856 670702 488960 951111 323046 446298 931161 31385 496951 264724 224916 169684

Formulating the problem

The limit of max possible weight is 13549094</td>
</tr>
<tr>
<td>TIME COMPLEXITY:</td>
<td> 
</td>
</tr>
</table>

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

	Time Complexity**: O (N*W).** where 'N' is the number of weight elements and 'W' is the capacity of the knapsack.

CONCLUSION: Learnt how to solve dynamic programming problems by dividing bigger problems into smaller problems.
Learnt about the time complexity of the Knapsack problem. Learnt how and why is the 2d array is used in the Knapack problem. Dynamic problems like Fibonacci series nth number uses array to store reccuring solution which is implemented here too.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	5		

AIM:	To implement Huffman encoding technique of greedy approach
THEORY	**What is Greedy Algorithm?** A greedy algorithm is a simple, intuitive algorithm that is used in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem. Greedy algorithms are quite successful in some problems, such as Huffman encoding which is used to compress data, or Dijkstra's algorithm, which is used to find the shortest path through a graph. **Structure of a Greedy Algorithm** Greedy algorithms take all of the data in a particular problem, and then set a rule for which elements to add to the solution at each step of the algorithm. In the animation above, the set of data is all of the numbers in the graph, and the rule was to select the largest number available at each level of the graph. The solution that the algorithm builds is the sum of all of those choices. If both of the properties below are true, a greedy algorithm can be used to solve the problem. • **Greedy choice property:** A global (overall) optimal solution can be reached by choosing the optimal choice at each step. • **Optimal substructure:** A problem has an optimal substructure if an optimal solution to the entire problem contains the optimal solutions to the sub-problems.

In other words, greedy algorithms work on problems for which it is true that, at every step, there is a choice that is optimal for the problem up to that step, and after the last step, the algorithm produces the optimal solution of the complete problem.
To make a greedy algorithm, identify an optimal substructure or subproblem in the problem. Then, determine what the solution will include (for example, the largest sum, the shortest path, etc.). Create some sort of iterative way to go through all of the subproblems and build a solution.

Limitations of Greedy Algorithms

Sometimes greedy algorithms fail to find the globally optimal solution because they do not consider all the data. The choice made by a greedy algorithm may depend on choices it has made so far, but it is not aware of future choices it could make.

Huffman Coding

Huffman encoding is another example of an algorithm where a greedy approach is successful. The Huffman algorithm analyzes a message and depending on the frequencies of the characters used in the message, it assigns a variable-length encoding for each symbol. A more commonly used symbol will have a shorter encoding while a rare symbol will have a longer encoding.

The Huffman coding algorithm takes in information about the frequencies or values of a particular symbol occurring. It begins to build the prefix tree from the bottom up, starting with the two least probable symbols in the list. It takes those symbols and forms a subtree containing them, and then removes the individual symbols from the list.

The algorithm sums the values of elements in a subtree and adds the subtree and its probability to the list. Next, the algorithm searches the list and selects the two symbols or subtrees with the smallest values. It uses those to make a new subtree, removes the original subtrees/symbols from the list, and then adds the

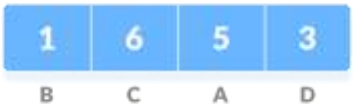
new subtree and its combined probability to the list. This repeats until there is one tree and all elements have been added. At each subtree, the optimal encoding for each symbol is created and together composes the overall optimal encoding.

Huffman Encoding Example:

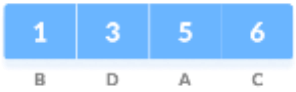
Suppose the string below is to be sent over a network.

Huffman coding is done with the help of the following steps.

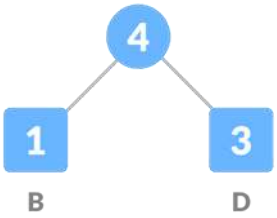
1. Calculate the frequency of each character in the string



2. Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.



3. Make each unique character as a leaf node.
4. Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.

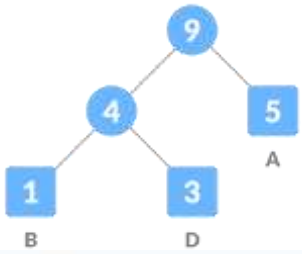


Computer Engineering Department &
Information Technology Engineering Department

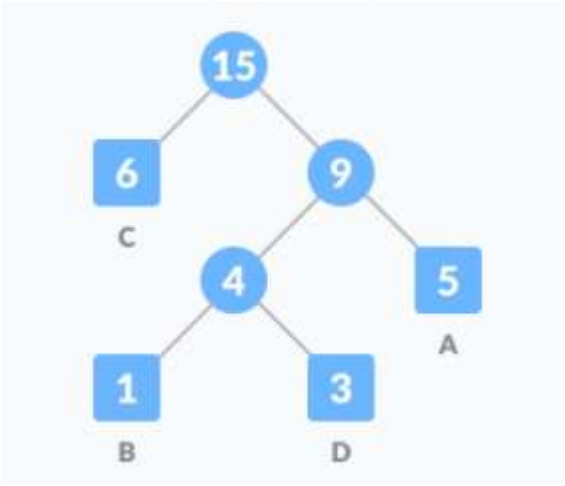
Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes in the figure above).
6. Insert node z into the tree.
7. Repeat steps 3 to 5 for all the characters



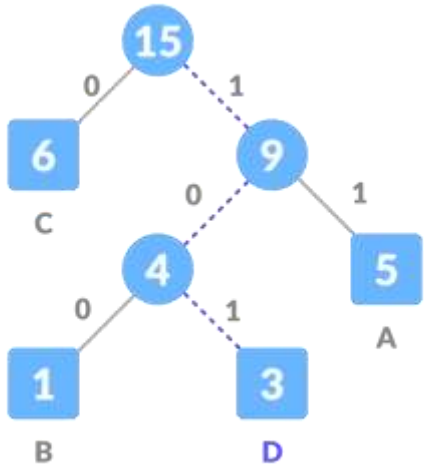
8. For each non-leaf node, assign 0 to the left edge and 1 to the right edge.



For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

Character	Frequency	Code	Size
A	5	11	5*2 = 10
B	1	100	1*3 = 3
C	6	0	6*1 = 6
D	3	101	3*3 = 9
4 * 8 = 32 bits	15 bits		28 bits

Decoding the code

	For decoding the code, we can take the code and traverse through the tree to find the character. Let 101 is to be decoded, we can traverse from the root as in the figure below. 
PSEUDOCODE:	**Huffman Algorithm** 1. Input:-Number of message with frequency count. 2. Output: - Huffman merge tree. 3. Begin 4. Let Q be the priority queue, 5. Q= {initialize priority queue with frequencies of all symbol or message} 6. Repeat n-1 times 7. Create a new node Z 8. X=extract_min(Q) 9. Y=extract_min(Q) 10. Frequency(Z) =Frequency(X) +Frequency(y); 11. Insert (Z, Q) 12. End repeat 13. Return (extract_min(Q)) 14. End.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	
CODE:	Huffman Code import java.util.*; class HuffmanNode { *// Huffman node class* int data; char character; *// constructor* HuffmanNode left, right; HuffmanNode() { this.left = null; this.right = null; } *// constructor* HuffmanNode(char ch, int data) { this.left = null; this.right = null; this.data = data; this.character = ch; } } class MyComparator implements Comparator<HuffmanNode> { *// Comparator class* public int compare(HuffmanNode x, HuffmanNode y) { return x.data - y.data; } } public class Huffman {

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
   // Huffman class

   char charArray[];
   int charFreq[];
   int characters;
   // User input
   Scanner input = new Scanner(System.in);
   // Huffman tree
   PriorityQueue<HuffmanNode> queue;
   // hashmap to store the encodings
   HashMap<Character, String> map = new
HashMap<Character, String>();

   // User input method
   public void userInput() {
      System.out.print("\nEnter the number of characters to
be read: ");
      characters = input.nextInt();

      // Priority queue to store the Huffman nodes
      queue = new
PriorityQueue<HuffmanNode>(characters, new
MyComparator());

      charArray = new char[characters];
      charFreq = new int[characters];

      System.out.print("\nEnter the characters below\n-
>");
      for (int i = 0; i < characters; i++) {
         charArray[i] = input.next().charAt(0);
      }

      System.out.print("\nEnter the Frequency of the
Characters\n");
      for (int i = 0; i < characters; i++) {
         System.out.print("-> '" + charArray[i] + "' : ");
         charFreq[i] = input.nextInt();
      }
```

```
   }

   public void printArrays() {
      System.out.println();
      System.out.print("\n|   Characters\t|" +
"   Frequency\t|\n");
      System.out.print("----------------------------------
\n");
      for (int i = 0; i < characters; i++) {
         System.out.print("\n|       " + charArray[i] + "\t|"
+ "        " + charFreq[i] + "\t|");
      }

   }

   public void setup(HashMap<Character, Integer> values)
{
      // setup method
      characters = values.size();
      charArray = new char[characters];
      charFreq = new int[characters];

      queue = new
PriorityQueue<HuffmanNode>(characters, new
MyComparator());

      for (int i = 0; i < values.size(); i++) {
         charArray[i] = (char) values.keySet().toArray()[i];
         charFreq[i] = (int) values.values().toArray()[i];
      }
      printArrays();
   }

   public HuffmanNode makeTree() {
      // makeTree method
      System.out.print("\nStarted Making the Huffman
Tree\n");
      for (int i = 0; i < characters; i++) {
```

```
            // Adding the nodes to the queue
            HuffmanNode hNode = new
HuffmanNode(charArray[i], charFreq[i]);
            queue.add(hNode);
        }
        HuffmanNode root = null;
        int counter = 1;
        // Making the tree
        while (queue.size() > 1) {
            System.out.print("\n---------------------------------
-\n");
            System.out.print("\nStep " + counter);
            HuffmanNode x = queue.peek();
            queue.poll();

            HuffmanNode y = queue.peek();
            queue.poll();

            HuffmanNode f = new HuffmanNode();
            // Combining the nodes
            f.data = x.data + y.data;
            f.character = '+';
            // Adding the combined node to the queue
            f.left = x;
            f.right = y;
            root = f;
            printNode(root, x, y);
            queue.add(f);
            counter++;
        }

        return root;
    }

    public void printNode(HuffmanNode root, HuffmanNode
x, HuffmanNode y) {
        System.out.print("\n\nParent Node: " + " " +
root.data);
        System.out.print("\n| |");
```

```
        System.out.printf("\n| \\_Left Child:\t|   " +
x.character + "\t |   " + x.data + " \t|");
        System.out.print("\n|");
        System.out.printf("\n\\_Right Child:\t|   " +
y.character + "\t |   " + y.data + " \t|");
    }

    public void printTree(HuffmanNode root, String
characters) {
        // System.out.println(root+" "+characters);
        if (root.left == null && root.right == null &&
Character.isLetter(root.character)) {
            map.put(root.character, characters);
            return;
        }
        // Printing the left child
        printTree(root.left, characters + "0");
        printTree(root.right, characters + "1");

    }

    public void displayMap() {
        // Displaying the map
        System.out.println("\n\nThe Encoding is: ");
        for (Map.Entry m : map.entrySet())
            System.out.println(m.getKey() + " " +
m.getValue());

    }
}



Driver Code
import java.util.HashMap;

public class Driver {

    public static void main(String[] args) {
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        Huffman HM = new Huffman();
        // user INput
        System.out.print("\n1.Character Input\n2.String
Input\nEnter your choice: ");
        int choice = HM.input.nextInt();
        switch (choice) {
            case 1:
                // character input
                HM.userInput();
                HM.printArrays();
                HuffmanNode root1 = HM.makeTree();
                HM.printTree(root1, "");
                HM.displayMap();
                break;
            case 2:
                // string input
                System.out.print("\nEnter the String to be
encoded(without space)");
                System.out.print("\n->");
                String str = HM.input.next();
                HashMap<Character, Integer> chars = new
HashMap<Character, Integer>();

                for (int i = 0; i < str.length(); i++) {
                    // System.out.print("\n" + str.charAt(i));
                    if
(Character.isLetter(str.toLowerCase().charAt(i)) &&
!chars.containsKey(str.charAt(i))) {
                        chars.put(str.toLowerCase().charAt(i), 1);
                    } else if (chars.containsKey(str.charAt(i))) {
                        int value = chars.get(str.charAt(i));
                        value++;
                        chars.replace(str.charAt(i), value);
                    }
                }
                System.out.print("\nAll the Characters in the
String are: ");
                HM.setup(chars);
```

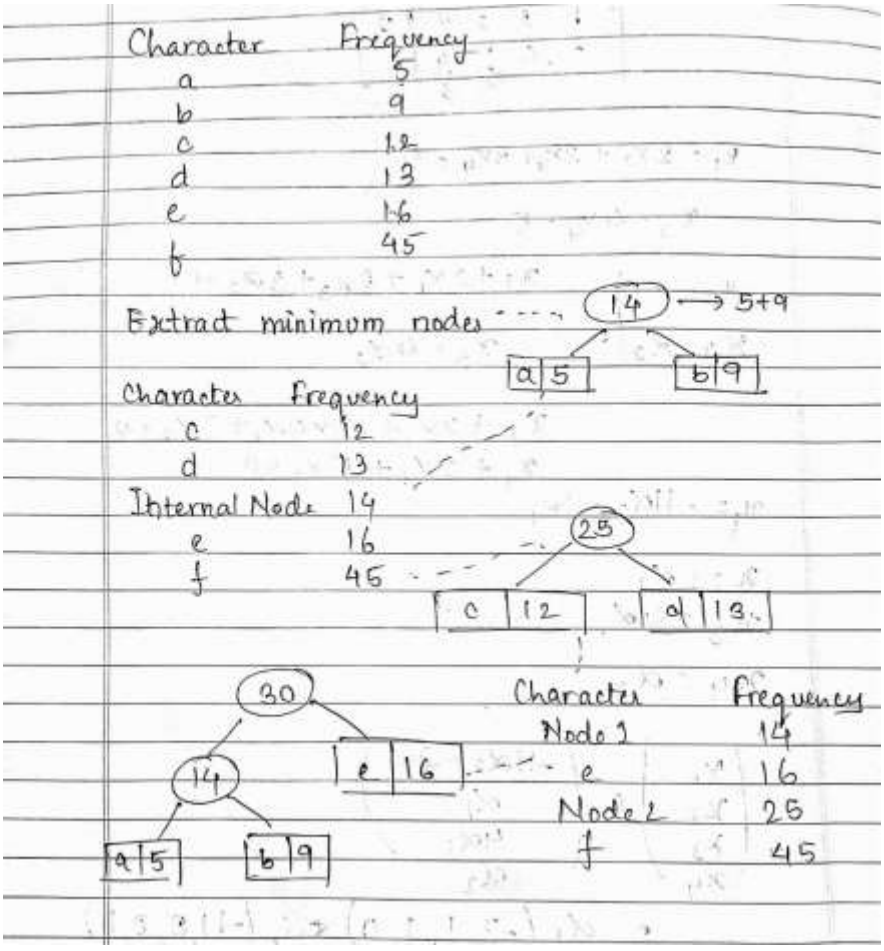
Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            HuffmanNode root2 = HM.makeTree();
            HM.printTree(root2, "");
            HM.displayMap();
            break;
         default:
            break;
      }
      // HM.printQueue();
   }
}
```

OUTPUT:

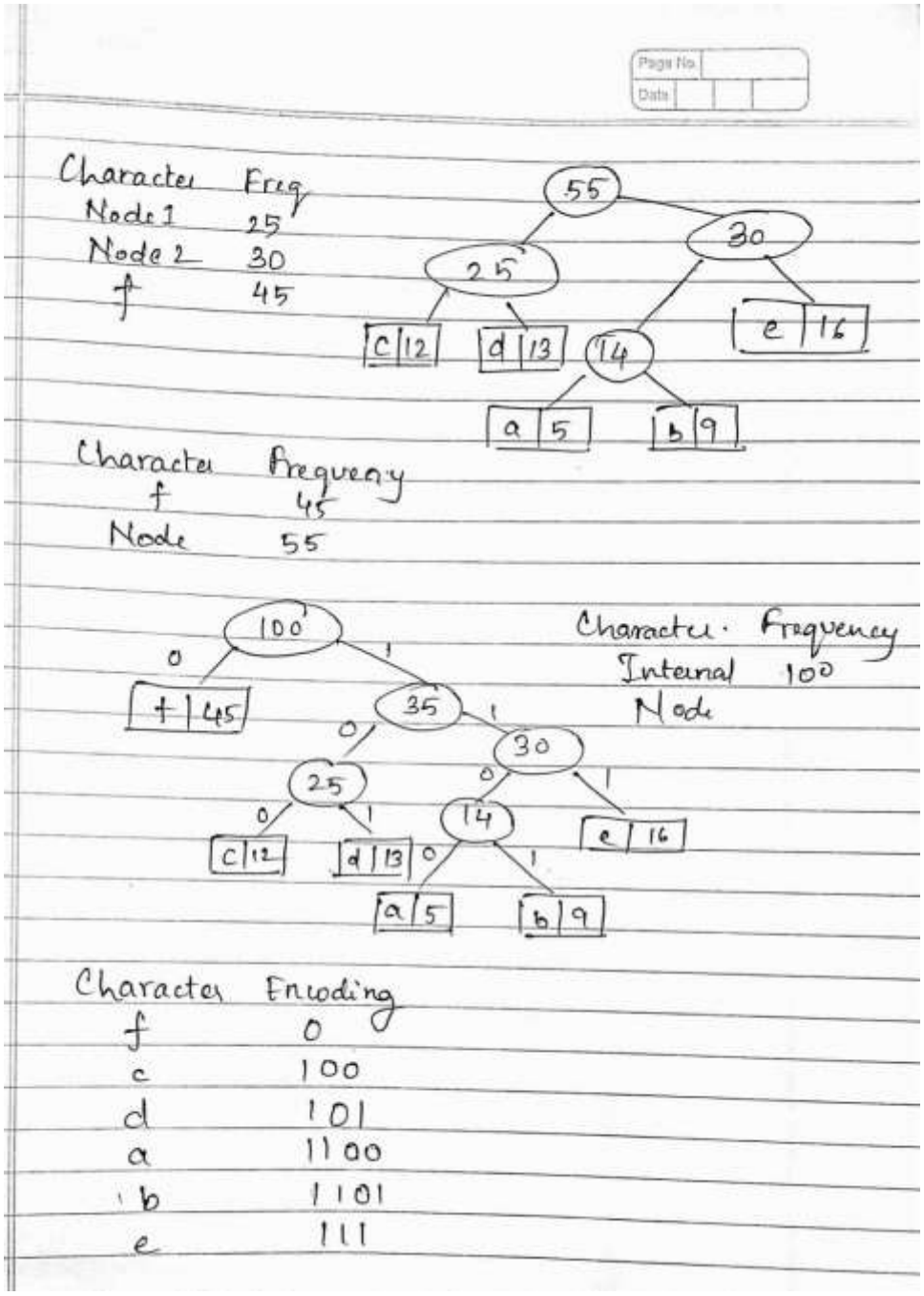


Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
1.Character Input
2.String Input
Enter your choice: 2

Enter the String to be encoded(without space)
->dawdawdnkladawdlawdkl

All the Characters in the String are:

|   Characters  |   Frequency   |
-----------------------------------

|        a      |          5    |
|        d      |          6    |
|        w      |          4    |
|        k      |          2    |
|        l      |          3    |
|        n      |          1    |
Started Making the Huffman Tree

-----------------------------------

Step 1

Parent Node:  3
| |
| \_Left Child: |   n    |   1  |
|
\_Right Child:  |   k    |   2  |
-----------------------------------

Step 2

Parent Node:  6
| |
| \_Left Child: |   l    |   3  |
|
\_Right Child:  |   +    |   3  |
-----------------------------------

Step 3

Parent Node:  9
| |
| \_Left Child: |   w    |   4  |
|
\_Right Child:  |   a    |   5  |
-----------------------------------
```

```
Step 4

Parent Node:  12
| |
| \_Left Child: |   +    |   6  |
|
\_Right Child:  |   d    |   6  |
-----------------------------------

Step 5

Parent Node:  21
| |
| \_Left Child: |   +    |   9  |
|
\_Right Child:  |   +    |   12        |

The Encoding is:
a 01
d 11
w 00
k 1011
l 100
n 1010
```

From the above example

- String is read and character is categorized according to the frequency
- Priority queue (min heap) is used to store the Huffman node
- Huffman tree is created using the queue
- Every node is printed with its child data
- In-order transversal through the tree goes through each node and assigns the encoding to it.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

TIME COMPLEXITY:	Operation of the Huffman algorithm. The time complexity of the Huffman algorithm is **O(nlogn)**. Using a heap to store the weight of each tree, each iteration requires **O(logn)** time to determine the cheapest weight and insert the new weight. There are **O(n)** iterations, one for each item.

RESULT: Things learnt during the procedural programming of the question

- Learnt about the Huffman encoding and decoding
- Learnt on how to store Huffman node priority queue (min heap)
- Learnt about the comparator class and used it in priority queue to find the min element
- Used priority queue which acts as a min heap to find the minimum most element in the heap
- Used HashMap to store the characters and frequency present in the string.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

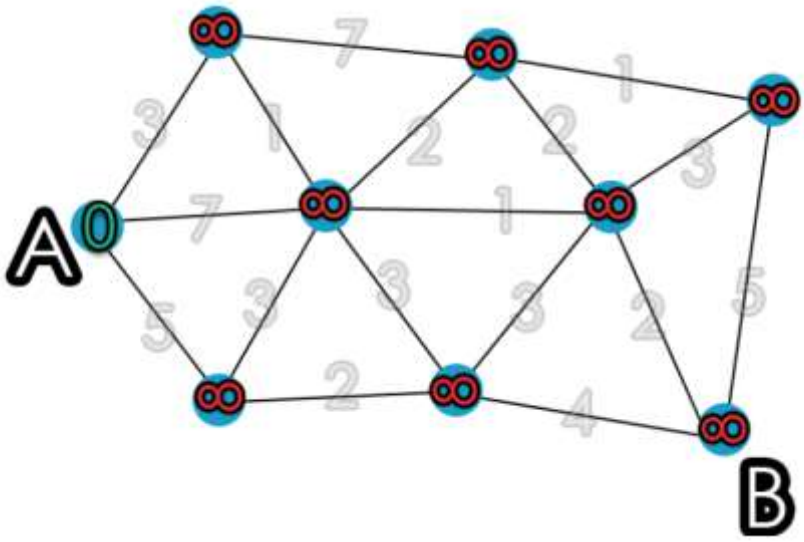
Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	6		

AIM:	To implement Dijkstra algorithm
THEORY:	**What is BackTracking?** Backtracking is an algorithmic technique where the goal is to get all solutions to a problem using the brute force approach. It consists of building a set of all the solutions incrementally. Since a problem would have constraints, the solutions that fail to satisfy them will be removed. It uses recursive calling to find a solution set by building a solution step by step, increasing levels with time. In order to find these solutions, a search tree named state-space tree is used. In a state-space tree, each branch is a variable, and each level represents a solution. A backtracking algorithm uses the depth-first search method. When it starts exploring the solutions, a bounding function is applied so that the algorithm can check if the so-far built solution satisfies the constraints. If it does, it continues searching. If it doesn't, the branch would be eliminated, and the algorithm goes back to the level before. **When to Use a Backtracking Algorithm** The backtracking algorithm is applied to some specific types of problems. For instance, we can use it to find a feasible solution to a decision problem. It was also found to be very effective for optimization problems. For some cases, a backtracking algorithm is used for the enumeration problem in order to find the set of all feasible solutions for the problem.

On the other hand, backtracking is not considered an optimized technique to solve a problem. It finds its application when the solution needed for a problem is not time-bounded.

Dijkstra's Algorithm

Dijkstra's algorithm finds a shortest path tree from a single source node, by building a set of nodes that have minimum distance from the source.



The graph has the following:

- vertices, or nodes, denoted in the algorithm by v*v* or u*u*;
- weighted edges that connect two nodes: (u,v*u*,*v*) denotes an edge, and w(u,v)*w*(*u*,*v*) denotes its weight. In the diagram on the right, the weight for each edge is written in gray.

This is done by initializing three values:

- dist*dist*, an array of distances from the source node s*s* to each node in the graph, initialized the following way: dist*dist*(s*s*) = 0; and for all other nodes v*v*, dist*dist*(v*v*) = \infty∞. This is done at the beginning because as the algorithm proceeds, the dist*dist* from the source to each node v*v* in the graph will be recalculated and finalized when the shortest distance to v*v* is found

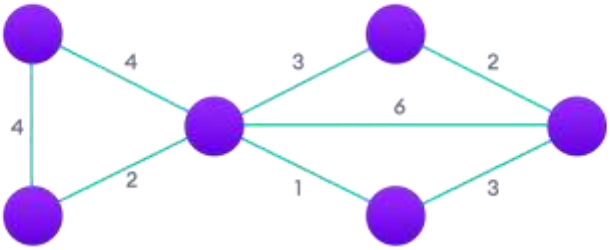
Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

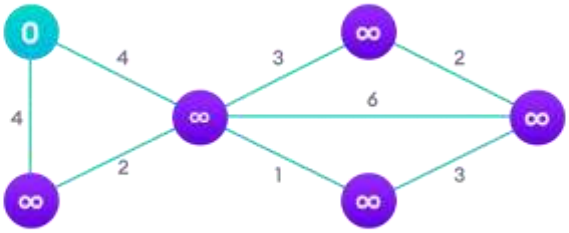
- Q*Q*, a queue of all nodes in the graph. At the end of the algorithm's progress, Q*Q* will be empty.
- S*S*, an empty set, to indicate which nodes the algorithm has visited. At the end of the algorithm's run, S*S* will contain all the nodes of the graph.

It is easier to start with an example and then think about the algorithm.



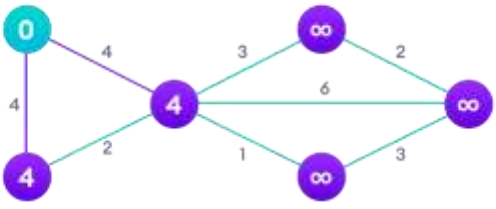
Step: 1

Start with a weighted graph



Step: 2

Choose a starting vertex and assign infinity path values to all other devices



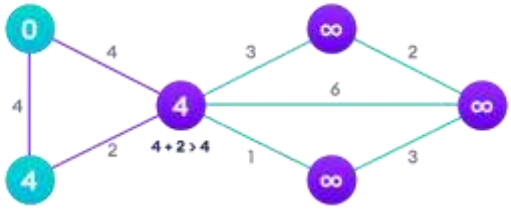
Step: 3

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

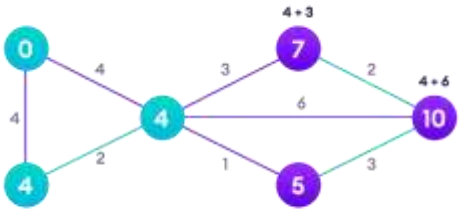
Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Go to each vertex and update its path length



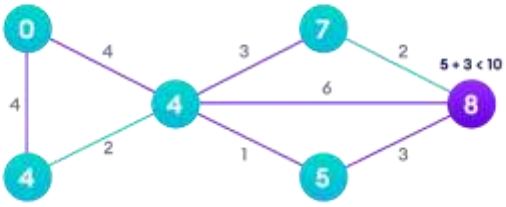
Step: 4

If the path length of the adjacent vertex is lesser than new path length, don't update it



Step: 5

Avoid updating path lengths of already visited vertices

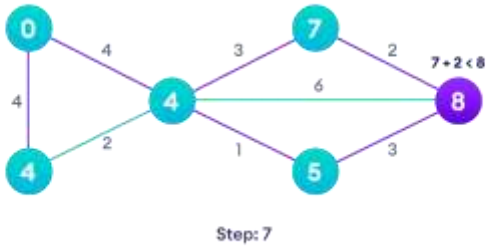


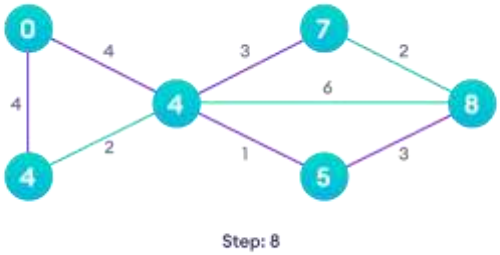
Step: 6

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

<table>
<tr>
<td></td>
<td>After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7
 

Step: 7
Notice how the rightmost vertex has its path length updated twice
 

Step: 8
Repeat until all the vertices have been visited</td>
</tr>
<tr>
<td>PSEUDOCODE:</td>
<td>1: function Dijkstra(Graph, source):
2: for each vertex v in Graph: // Initialization
3: dist[v] := infinity // initial distance from source to vertex v is set to infinite
4: previous[v] := undefined // Previous node in optimal path from source
5: dist[source] := 0 // Distance from source to source
6: Q := the set of all nodes in Graph // all nodes in the graph are unoptimized - thus are in Q
7: while Q is not empty: // main loop
8: u := node in Q with smallest dist[ ]
9: remove u from Q
10: for each neighbor v of u: // where v has not yet been removed from Q.
11: alt := dist[u] + dist_between(u, v)
12: if alt < dist[v] // Relax (u,v)</td>
</tr>
</table>

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
13:   dist[v] := alt
14:   previous[v] := u
15:   return previous[ ]
```

EXPERIMENT 1

CODE:

```
import java.util.*;

public class dijkstra {

   int source;

   // The main method is where the program starts.
   void dijkstra_solve(int[][] graph) {
      int count = graph.length;
      boolean[] visited = new boolean[count];
      int[] distance = new int[count];

      for (int i = 0; i < distance.length; i++) {
         distance[i] = Integer.MAX_VALUE;
      }
      distance[source] = 0;
      for (int k = 0; k < distance.length - 1; k++) {

         int minVertex = findMin(distance, visited);
         visited[minVertex] = true;
         // explore the neighbours
         for (int i = 0; i < distance.length; i++) {
            if (graph[minVertex][i] != 0 &&
distance[minVertex] != Integer.MAX_VALUE) {
               // checking if the there exists an edge
between the two vertices, the neighbour
               // should not be visited
               // adding the weight of the edge to the
distance of the min vertex
               int newDistance = distance[minVertex] +
graph[minVertex][i]; // Relaxation
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
                if (newDistance < distance[i]) {
                    // updating the distance of the vertex if the
value is lesser than the
                    // previous value of the same vertex
                    distance[i] = newDistance;
                }
            }
        }
    }
    System.out.println("\nOutput\n(Vertex-> Distance):");
    for (int i = 0; i < distance.length; i++) {
        if (distance[i] == Integer.MAX_VALUE || distance[i]
== 0) {
            continue;
        }
        System.out.println(i + "\t" + distance[i]);
    }
}

// finding the minimum distance vertex
static int findMin(int[] distance, boolean[] visited) {
    int minVertex = -1; // initializing the minimum vertex
to -1
    for (int i = 1; i < distance.length; i++) {
        // if the vertex is not visited and the distance is
lesser than the min vertex
        if ((minVertex == -1 || distance[i] <
distance[minVertex]) && !visited[i]) {
            minVertex = i;
        }
    }
    return minVertex; // returning the minimum vertex
}

public static void main(String[] args) throws Exception {
    try (// Driver code
        Scanner sc = new Scanner(System.in)) {
        System.out.println("----------------Dijkstra's
Algorithm----------------");
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        System.out.println("\nInput(TestCases-> Vertices-> Edges-> Each edge with weights)\n");
        int testCases = sc.nextInt();
        dijkstra T = new dijkstra();
        int negativeChecker = 0;
        for (int i = 0; i < testCases; i++) {
            int v = sc.nextInt(); // vertices
            int e = sc.nextInt(); // edges
            int[][] graph = new int[1024][1024]; // adjacency matrix
            int src = sc.nextInt();
            T.source = src;
            int dest = sc.nextInt();
            int cost = sc.nextInt(); // cost of the edge // set to store the vertices
            // if cost is negative, then the edge is not added
            graph[src][dest] = cost;
            if (cost < 0) {
                negativeChecker = 1;
                System.out.print("\nNegative edge not Added");
                continue;
            }
            for (int j = 0; j < e - 1; j++) {
                int p = sc.nextInt(); // source
                int q = sc.nextInt(); // destination
                cost = sc.nextInt(); // cost of the edge
                if (cost < 0) {
                    negativeChecker = 1;
                    System.out.print("\nNegative edge not allowed");
                    break;
                }

                graph[p][q] = cost;
            }
            if (negativeChecker != 1)
                T.dijkstra_solve(graph);
        }
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            sc.close();
        }
    }
}
```

OUTPUT:

```
Input(TestCases-> Vertices-> Edges-> Each e

1
3 3
5 7 1
6 5 2
7 6 3

Output
(Vertex-> Distance):
6       4
7       1
```

```
1
6 5
1 2 15
1 3 5
3 2 6
2 4 2
5 6 7

Output
(Vertex-> Distance):
2       11
3       5
4       13
```

```
1
6 7
10 20 3
20 40 -1
40 4 -2
4 80 1
4 2 -2
80 2 -4

Negative edge not allowed
```

```
1
7 7
15 3 2
15 19 5
3 14 3
19 8 2
19 10 4
14 8 1
8 16 5

Output
(Vertex-> Distance):
3       2
8       6
10      9
14      5
16      11
19      5
```

```
1
8 13
8 7 4
8 6 4
5 8 3
5 6 2
6 3 -2
6 4 4
3 7 3
3 4 -3
4 5 1
4 2 -2
1 4 -2
2 1 2

Negative edge not allowed
```

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
Output
(Vertex-> Distance):
1       461
2       417
3       443
4       522
5       498
6       427
7       441
8       207
9       476
10      437
11      549
12      481
13      448
14      348
15      397
16      469
17      409
18      438
19      473
20      437
21      515
23      430
24      518
25      461
26      533
27      397
28      392
29      481
32      462
33      386
34      349
35      399
36      425
37      474
38      384
39      444
40      470
41      442
43      391
44      443
45      412
46      487
47      413
48      429
49      435
50      411
51      506
52      434
53      489
54      493
55      474
56      491
57      403
58      462
60      434
61      490
62      471
64      477
66      457
67      96
68      452
69      453
70      431
71      388
72      418
73      446
74      494
75      126
76      499
77      354
78      342
79      476
80      607
81      436
82      456
83      461
84      519
86      394
87      396
88      507
89      477
90      467
91      382
92      460
93      414
94      432
95      201
96      345
97      496
98      424
99      420
100     446
101     524
102     390
103     418
104     459
105     499
106     466
107     482
108     460
109     472
110     449
111     428
112     516
113     421
114     288
115     457
116     494
117     555
118     515
119     494
120     483
121     408
122     465
123     391
124     461
125     610
126     450
127     457
128     399
129     473
130     473
131     460
132     414
133     418
134     408
135     607
136     429
137     442
138     424
139     461
140     624
141     492
142     355
144     436
145     343
146     412
147     433
148     445
149     389
150     393
152     448
153     576
154     447
155     474
157     448
158     408
159     478
160     381
161     412
162     438
163     453
164     508
165     465
166     392
167     492
168     492
169     456
170     416
171     525
172     459
173     429
174     452
175     441
176     409
177     479
178     476
179     399
180     502
181     400
183     545
184     557
185     390
186     536
187     467
188     465
189     438
190     495
191     404
192     400
193     469
195     459
196     369
197     447
198     480
199     524
200     413
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

201	477	246	436
202	623	247	380
203	468	249	532
204	312	250	356
205	384	251	39
206	531	252	443
207	515	253	491
209	468	254	456
210	371	255	513
211	420	256	441
212	415	257	409
213	355	258	497
214	485	259	449
217	509	260	440
218	511	261	447
219	438	262	498
220	464	263	451
221	385	264	456
222	546	265	472
223	461	266	361
224	527	267	383
226	477	268	476
227	517	269	395
228	432	270	530
229	435	271	427
230	454	272	528
232	443	274	393
233	442	275	402
234	471	277	490
235	483	278	415
236	468	279	513
237	490	280	446
238	475	281	522
240	386	282	493
241	394	283	444
242	498	284	426
244	483	285	451
245	464	286	412
		287	503

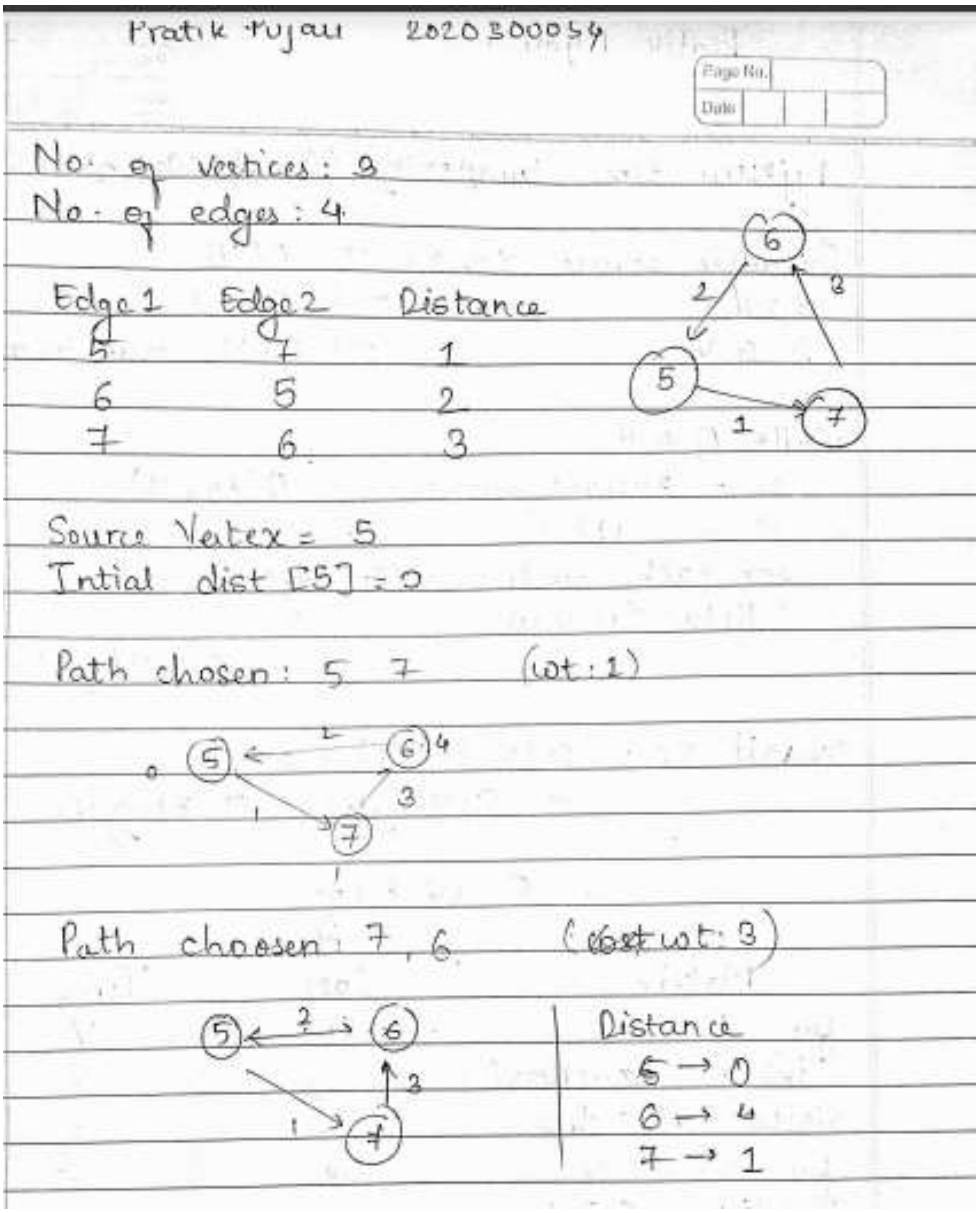
288	444
289	463
291	394
292	452
293	514
294	431
295	552
296	432
297	401

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

	Written Sum for the Dijkstra 

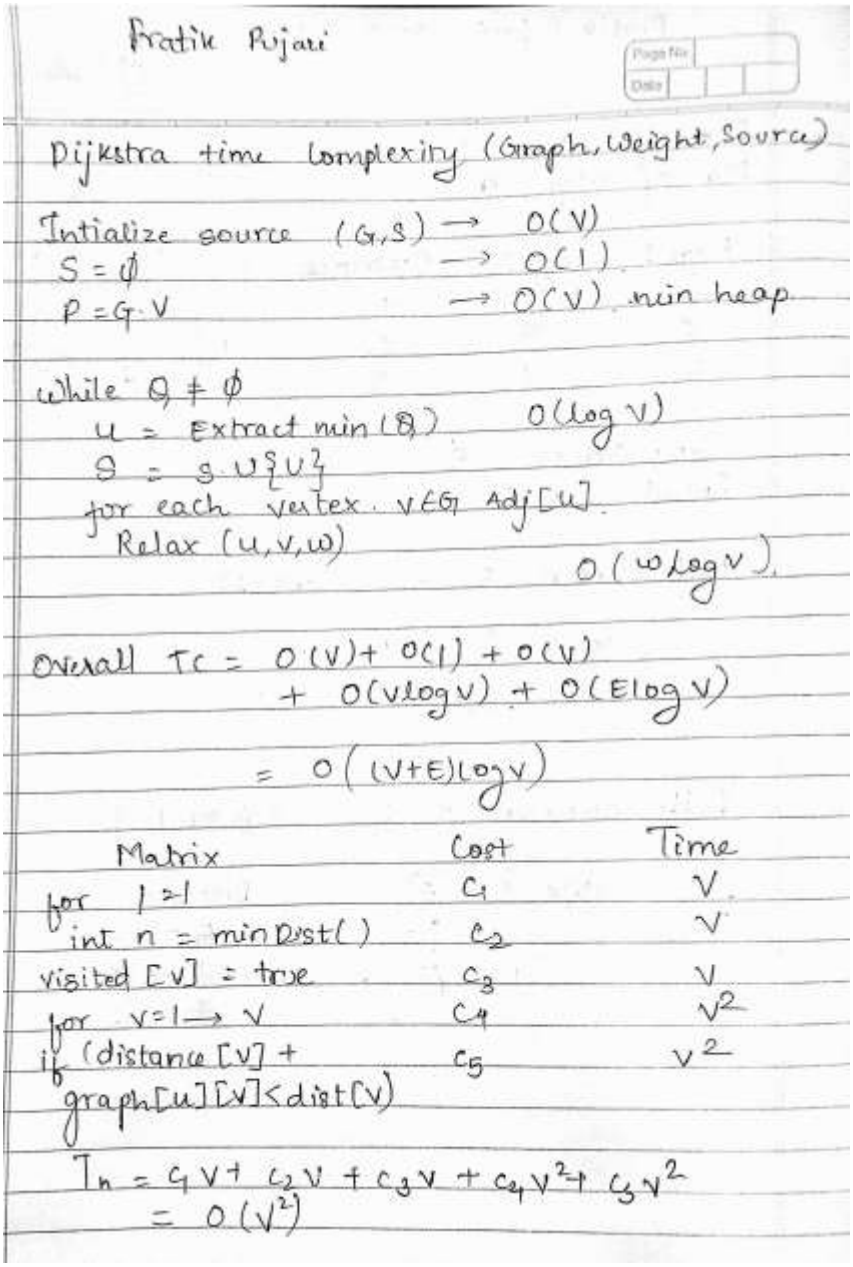
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

TIME COMPLEXITY:



Pratik Rajani

Dijkstra time complexity (Graph, Weight, Source)

Intialize source (G,S) → O(V)
S = ∅ → O(1)
P = G.V → O(V) min heap

while Q ≠ ∅
u = Extract min (Q) O(log V)
S = S ∪ {u}
for each vertex v ∈ G Adj[u]
Relax (u,v,w)
O(w log v).

Overall Tc = O(V) + O(1) + O(V)
+ O(V log V) + O(E log V)
= O((V+E) log V)

Matrix	Cost	Time
for i=1	c_1	V
int n = minDist()	c_2	V
visited[v] = true	c_3	V
for v=1 → V	c_4	V^2
if (distance[v] + graph[u][v] < dist[v])	c_5	V^2

$T_n = c_1V + c_2V + c_3V + c_4V^2 + c_5V^2$
$= O(V^2)$

Both the time and space complexity obtained for the Dijkstra's algorithm is O(V*V) where V is the number of vertices. While if an adjacency list is used the time complexity will come out to be O(E*logV) where E is the number of edges. Since the Dijkstra's algorithm works on

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

	Greedy approach, it has a limitation that it doesn't work for the negative values of the distances because negative value and since Dijkstra works on the principle that once a vertex is discovered, it can't go back. Thus, it can't be used for the negative values

CONCLUSION: Learnt during the procedural programming of solving the problem

- Learnt about BackTracking and Dijsktra Algorithm
- Learnt how to use adjacency matrix in order to store the graph
- Learnt about time and space complexity of the shortest path algorithm
- Learnt about the advantages and disadvantages of dijsktra algorithm
- Learnt about the applications of a Dijsktra/ shortest path finder algorithm

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

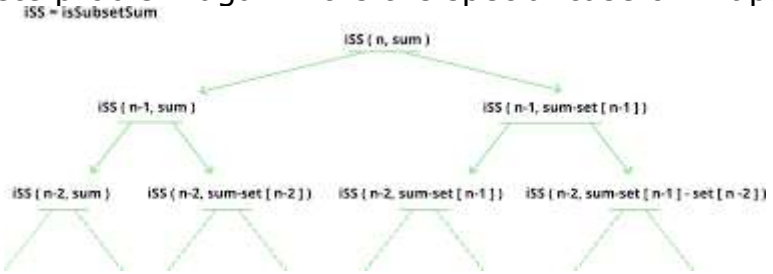
Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	7		

AIM:	To implement the concept of backtracking in subset sum problem
THEORY:	**What is Backtracking?** Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree). There are three types of problems in backtracking – 1. Decision Problem – In this, we search for a feasible solution. 2. Optimization Problem – In this, we search for the best solution. 3. Enumeration Problem – In this, we find all feasible solutions. **When to Use a Backtracking Algorithm** The backtracking algorithm is applied to some specific types of problems. For instance, we can use it to find a feasible solution to a decision problem. It was also found to be very effective for optimization problems. For some cases, a backtracking algorithm is used for the enumeration problem in order to find the set of all feasible solutions for the problem. On the other hand, backtracking is not considered an optimized technique to solve a problem. It finds its

application when the solution needed for a problem is not time-bounded.

Subset Sum Problem

It is one of the most important problems in complexity theory. The problem is given an A set of integers a1, a2,…., an upto n integers. The question arises that is there a non-empty subset such that the sum of the subset is given as M integer?. For example, the set is given as [5, 2, 1, 3, 9], and the sum of the subset is 9; the answer is YES as the sum of the subset [5, 3, 1] is equal to 9. This is an NP-complete problem again. It is the special case of knapsack



Example

Consider the following array/ list of integers:
{1, 3, 2}
We want to find if there is a subset with sum 3.
Note that there are two such subsets **{1, 2}** and **{3}**. We will follow our backtracking approach.
Consider our empty set **{}**
We add 1 to it **{1}** (sum = 1, 1 < 3)
We add 2 to it **{1, 3}** (sum = 3, 3 == 3, found)
We remove 3 from it **{1}** (sum = 1, 1 < 3)
We add 2 to it **{1, 2}** (sum = 3, 3 == 3, found)
We remove 2 and see that all elements have been considered.
Following diagram captures the idea:

{} -> {1} -> {1, 3} (found) -> {1} -> {1, 2} (found) -> {1} (end)

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

PSEUDOCODE:	subset_sum() if(subset is satisfying the constraint) print the subset exclude the current element and consider next element else generate the nodes of present level along breadth of tree and recur for next levels
EXPERIMENT 1	
CODE:	#include <stdio.h> #include <stdlib.h> static int total_nodes; void printValues(int A[], int size){ //prints the array printf("\n\n"); for (int i = 0; i < size; i++) { printf("%*d", 5, A[i]); } } void subset_sum(int s[], int t[], int s_size, int t_size, int sum, int ite, int const target_sum){ //increments the total node count total_nodes++; if (target_sum == sum) { // target sum found and printing the list. printValues(t, t_size); //finding all other valid pairs subset_sum(s, t, s_size, t_size - 1, sum - s[ite], ite + 1, target_sum); return; } else { //checking for all possible combinations for (int i = ite; i < s_size; i++) { t[t_size] = s[i];

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        subset_sum(s, t, s_size, t_size + 1, sum
+ s[i], i + 1, target_sum);
      }
   }
}
void generateSubsets(int s[], int size, int target_sum){
//generating subsets of the initial array
   int* tuplet_vector = (int*)malloc(size * sizeof(int));
   subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
   free(tuplet_vector);
}
int main(){
   int size;
   int target_sum;
//user input like array sze and array and target sum
   printf("\n-------------------------------");
   printf("\nEnter the size of the set: ");
   scanf("%d", &size);
   int set[size];
   printf("\n-------------------------------");
   printf("\nEnter the elements of the set: ");
   for (int i = 0; i < size; i++) {
      scanf("%d", &set[i]);
   }
   printf("The set is ");
   printValues(set , size);
   printf("\n-------------------------------");
   printf("\nEnter the target sum: ");
   scanf("%d", &target_sum);

//calling the functions and getting the result
   generateSubsets(set, size, target_sum);
//printing the total nodes.
   printf("\n\nTotal Nodes generated %d\n", total_nodes);
   return 0;
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

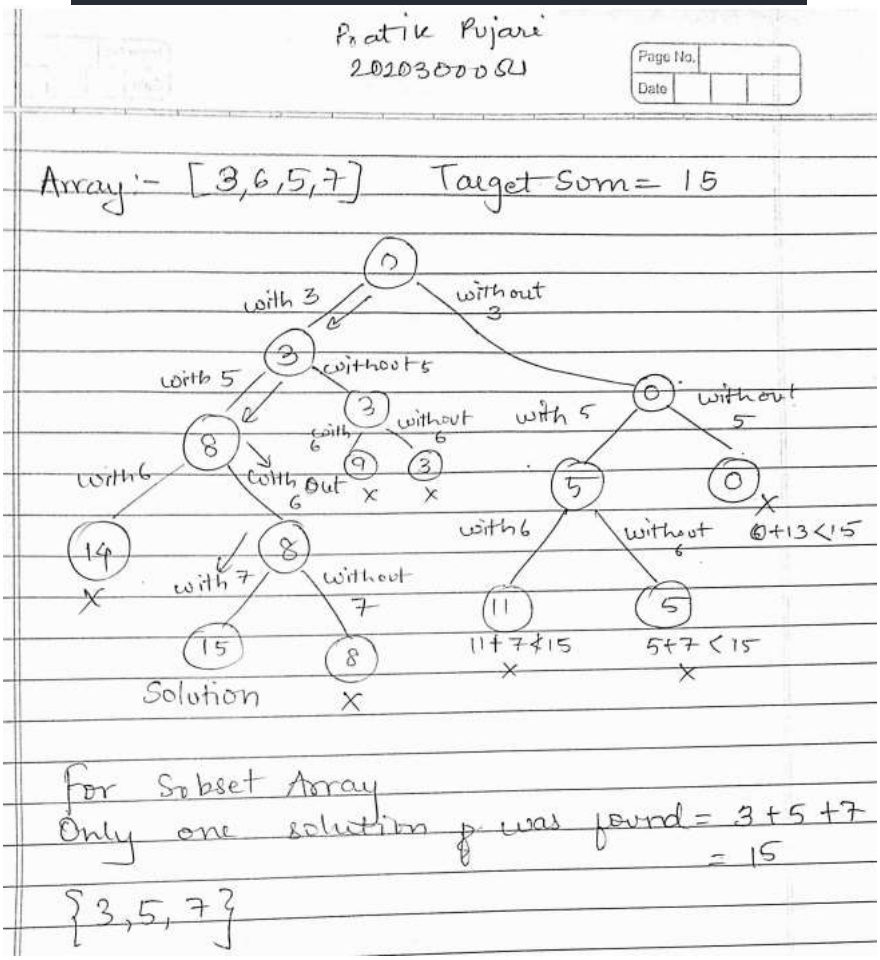
Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

OUTPUT:

```
--------------------------------
Enter the size of the set: 6

--------------------------------
Enter the elements of the set: 12 1 61 5 9 2
The set is

   12    1   61    5    9    2
--------------------------------
Enter the target sum: 24


   12    1    9    2

Total Nodes generated 65
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

TIME COMPLEXITY:	The time complexity of the sum of subset problem is O(2^n) using backtracking approach ,where n is the number of elements in the array Since there will be two branches of each node and every element is atleast traversed once making it a binary tree. The total number of nodes can be calculated by the formula 2^n-1 where is the number of elements in the array.

CONCLUSION: Things learnt during the procedural solving of the program.

- Learnt how to use backtracking for solving sums that use a binary tree structure.
- Learnt how to analyse the time complexity to find the number of nodes in diagram
- Learnt to implement backtracking solution to the subset problem.
- Learnt how to solve subset problem using the backtracking method using the tree way.

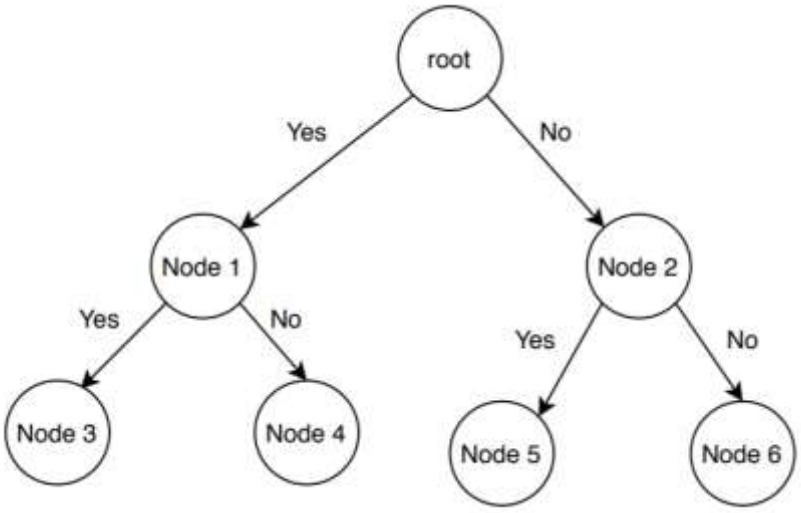
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	8		

AIM:	To implement 15 puzzle sum using the branch and bound
THEORY:	**What is Branch and Bound?** Branch and bound algorithms are used to find the optimal solution for combinatory, discrete, and general mathematical optimization problems**.** In general, given an NP-Hard problem, a branch and bound algorithm explores the entire search space of possible solutions and provides an optimal solution. A branch and bound algorithm consist of stepwise enumeration of possible candidate solutions by exploring the entire search space. With all the possible solutions, we first build a rooted decision tree. The root node represents the entire search space:  **Advantages**

In a branch and bound algorithm, we don't explore all the nodes in the tree. That's why the time complexity of the branch and bound algorithm is less when compared with other algorithms.
If the problem is not large and if we can do the branching in a reasonable amount of time, it finds an optimal solution for a given problem.
The branch and bound algorithm find a minimal path to reach the optimal solution for a given problem. It doesn't repeat nodes while exploring the tree.

Disadvantages

The branch and bound algorithm are time-consuming. Depending on the size of the given problem, the number of nodes in the tree can be too large in the worst case.
Also, parallelization is extremely difficult in the branch and bound algorithm.

Lets solve an example for branch and bound for 15 puzzle problem

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

Initial arrangement

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

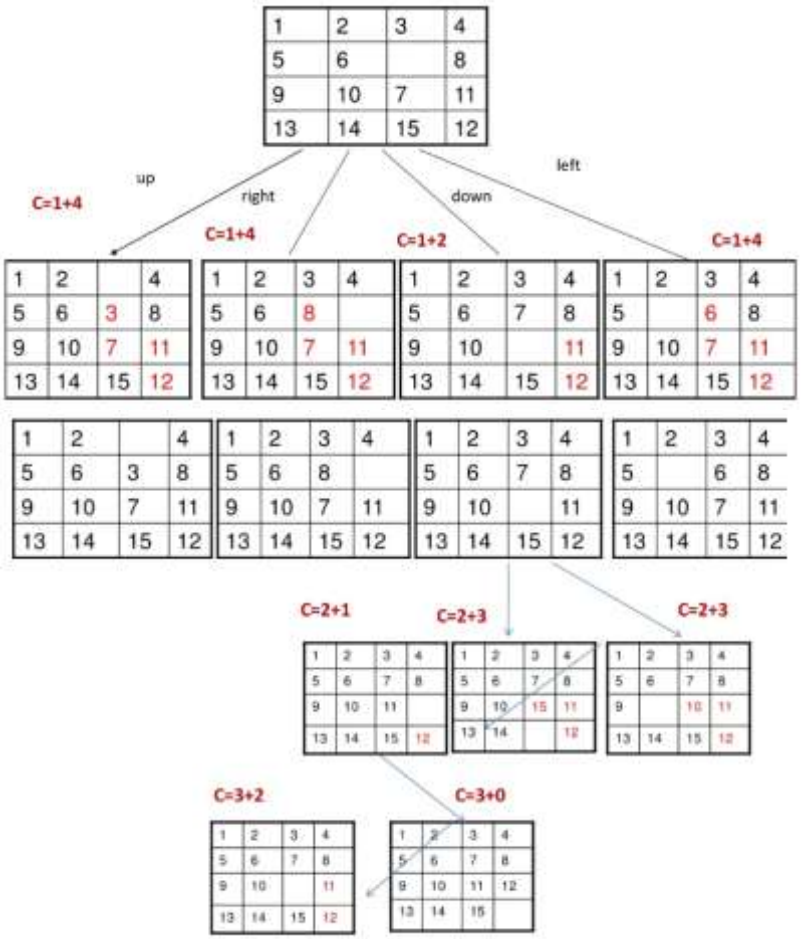
Goal arrangement

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



PSEUDOCODE:

```
struct list_node
{
list_node *next;
list_node *parent;
float cost;
};
algorithm LCSearch(list_node *t)
{
        if (*t is an answer node)
      {
         print(*t);
         return;
}
        E =t;
Initialize the list of live nodes to be empty;
while (true) {
for each child x of E
{
      if x is an answer node {
        print the path from x to t;
        return; }
      Add (x);
      x->parent = E; }
     if there are no more live nodes
   {
        print ("No answer node");
        return;
   }
      E = Least();
   }
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	
CODE:	import java.util.*; class Node { int[][] array; int misplaced; int recent; *// 1 up| 2 right| 3 down| 4 left|* } class branchBounds { int blankRow; int blankRowIndex;*// row index* int blackColIndex;*// column index* int totalCost; String isChoosen = "None"; int[][] targetMatrix = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 0 } }; boolean isSolvable(int[][] arr, int row_len) { int inv = inversions(arr); *// method that counts the number of inversions (i<j, arr[i]>arr[j])* if (arr.length % 2 != 0) {*// checks if n is odd* if (inv % 2 == 0) {*// if the length is odd, and inversions are even the puzzle is solvable* return true; } *// row: even AND inversion: odd =>solvable* *// row: odd AND inversion: even =>solvable* } else {*// if n is even* if (this.blankRow % 2 == 0 && inv % 2 != 0 || this.blankRow % 2 != 0 && inv % 2 == 0) { return true; } } return false;

```
    }

    int inversions(int[][] arr) {
        // count the number of inversions
        int no_inversions = 0;
        int[] arr2 = new int[arr.length * arr.length]; // 1d
array
        int k = 0;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                arr2[k] = arr[i][j]; // converting 2d array into 1d
array
                if (arr[i][j] == 0) {// blank tile(finding the index
of the blank tile)
                    this.blankRow = arr.length - i; // findimg the
index according to the convention (bottom->top):
                                                    // 1..2 3.
                    this.blankRowIndex = i;
                    this.blackColIndex = j;
                }
                k++;
            }
        }
        printArray(arr);
        System.out.println();
        System.out.println("----------------------");
        System.out.println("X Mark is at -> " +
(blankRowIndex + 1) + ", " + (blackColIndex + 1));
        System.out.println("----------------------");

        System.out.println();
        for (int i = 0; i < arr2.length; i++) {
            for (int j = i + 1; j < arr2.length; j++) {
                if (arr2[i] > arr2[j] && arr2[j] != 0) { // not
considering the blank tile while finding out the
                    // inversions
                    no_inversions++;
                }
            }
```

```
        }
        System.out.println("Total inversions: " +
no_inversions);
        return no_inversions;
    }

    boolean isMatched(int[][] arr, int[][] sel) {
        // check if the selected array is the target array
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                if (arr[i][j] != sel[i][j]) {// checks the array with
the target array, as soon as it matches the while
                    // loop exits
                    return false;
                }
            }
        }
        return true;
    }

    int mismatch(int[][] arr) {
        // misplaced tiles
        int mislocations = 0;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                if (arr[i][j] != this.targetMatrix[i][j] && arr[i][j]
!= 0) {// checks the number of elements that dont
                                                                //
match the target array
                    mislocations++;
                }
            }
        }
        return mislocations;
    }

    void solve(int[][] arr) {
        // Solving the puzzle
        int cost = Integer.MAX_VALUE;
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        int level = 0;
        int[][] temp_array = new int[arr.length][arr.length];

        while (!isMatched(arr, targetMatrix)) {
            level++;
            for (int i = 0; i < arr.length; i++) {
                for (int j = 0; j < arr.length; j++) {
                    // blank tile index
                    // checking where the x mark is
                    if (arr[i][j] == 0) {
                        this.blankRow = arr.length - i;
                        this.blankRowIndex = i;
                        this.blackColIndex = j;
                    }

                }
            }
            System.out.print("\nCosts->\n");
            int left[][] = leftShift(arr, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Left shift: " + ((int)
mismatch(left) + (int) level));

            int up[][] = upShift(arr, left, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Up shift: " + ((int)
mismatch(up) + (int) level));

            int right[][] = rightShift(arr, up, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Right shift: " + ((int)
mismatch(right) + (int) level));

            int[][] down = downShift(arr, right, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Down shift: " + ((int)
mismatch(down) + (int) level));
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        for (int i = 0; i < down.length; i++) { // storing the array for down shift
            for (int j = 0; j < down.length; j++) {
                down[i][j] = arr[i][j];
            }
        }

        if (blankRowIndex != arr.length - 1) {// checks if the down shift is possible and doesnt go out of bounds
            int temp = down[blankRowIndex + 1][blackColIndex];
            down[blankRowIndex + 1][blackColIndex] = down[blankRowIndex][blackColIndex];
            down[blankRowIndex][blackColIndex] = temp;
        }

        if (mismatch(down) + level <= cost) {// checking if the cost is lower
            cost = mismatch(down) + level;
            for (int i = 0; i < left.length; i++) {
                for (int j = 0; j < left.length; j++) {
                    temp_array[i][j] = down[i][j];
                }
            }

        }
        System.out.print("\nMinimum possible cost: " + ((int) mismatch(down) + (int) level) + "\n");

        System.out.print("\nOperation performed: " + isChoosen + "\n\n");

        // after filtering through the whole level printing the current
        for (int i = 0; i < down.length; i++) {
            for (int j = 0; j < down.length; j++) {
                arr[i][j] = temp_array[i][j];
                // status of the matrix
            }
```

```
            }
            printArray(arr);

            totalCost = totalCost + cost;

        }
        System.out.println("Total cost: " + totalCost);
    }

    public void printArray(int arr[][]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.println("----------------------------");
            for (int j = 0; j < arr.length; j++) {
                System.out.print(String.format("| %3d  ",
arr[i][j]));
            }
            System.out.println("|");
        }
        System.out.println("----------------------------");
    }

    public int[][] leftShift(int[][] arr, int[][] temp_array, int
blankRowIndex, int blackColIndex, int level,
            int cost) {
        // left shift
        int[][] left = new int[arr.length][arr.length];
        // storing the array for left shift
        for (int i = 0; i < left.length; i++) {
            for (int j = 0; j < left.length; j++) {
                left[i][j] = arr[i][j];
            }
        }
        // checks if the left shift is possible and doesnt go out
of bounds
        if (blackColIndex != 0) {
            int temp = left[blankRowIndex][blackColIndex];
            left[blankRowIndex][blackColIndex] =
left[blankRowIndex][blackColIndex - 1];
            left[blankRowIndex][blackColIndex - 1] = temp;
```

```
        }
        // checking if the cost is minimum
        if (mismatch(left) + level <= cost) {
            isChoosen = "Shifting left";
            cost = mismatch(left) + level; // assigning lower
cost
            for (int i = 0; i < left.length; i++) {
                for (int j = 0; j < left.length; j++) {
                    temp_array[i][j] = left[i][j]; // potential
candidate
                }
            }
        }
        return temp_array;
    }

    public int[][] rightShift(int[][] arr, int[][] up, int[][]
temp_array, int blankRowIndex, int blackColIndex,
            int level,
            int cost) {
        // right shift
        int[][] right = new int[arr.length][arr.length];
        // storing the array for right shift
        for (int i = 0; i < right.length; i++) {
            for (int j = 0; j < right.length; j++) {
                right[i][j] = arr[i][j];
            }
        }
        // checks if the right shift is possible and doesnt go
out of bounds
        if (blackColIndex != arr.length - 1) {
            int temp = right[blankRowIndex][blackColIndex];
            right[blankRowIndex][blackColIndex] =
right[blankRowIndex][blackColIndex + 1];
            right[blankRowIndex][blackColIndex + 1] = temp;
        }
        // checking if the cost is minimum
        if (mismatch(right) + level <= cost) {
            isChoosen = "Shifting right";
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            cost = mismatch(right) + level; // assigning lower
cost
            for (int i = 0; i < right.length; i++) {
               for (int j = 0; j < right.length; j++) {
                  temp_array[i][j] = right[i][j]; // potential
candidate
               }
            }
         }
         return temp_array;
      }

      public int[][] upShift(int arr[][], int left[][], int
temp_array[][], int blankRowIndex, int blankColIndex,
            int level,
            int cost) {
         int[][] up = new int[arr.length][arr.length];

         for (int i = 0; i < up.length; i++) {
            for (int j = 0; j < up.length; j++) {
               up[i][j] = arr[i][j]; // storing the array for up
shift
            }
         }
         if (blankRowIndex != 0) { // checks if the up shift is
possible and doesn't go out of bounds

            int temp = up[blankRowIndex - 1][blankColIndex];
            up[blankRowIndex - 1][blankColIndex] =
up[blankRowIndex][blankColIndex];
            up[blankRowIndex][blankColIndex] = temp;
         }

         if (mismatch(up) + level <= cost) { // checking if the
cost is lower
            isChoosen = "Shifting Up";
            cost = mismatch(up) + level;
            for (int i = 0; i < left.length; i++) {
               for (int j = 0; j < left.length; j++) {
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
                temp_array[i][j] = up[i][j];
            }
        }
    }
    return temp_array;
}

public int[][] downShift(int arr[][], int left[][], int
temp_array[][], int blankRowIndex, int blankColIndex,
        int level,
        int cost) {
    int[][] down = new int[arr.length][arr.length];

    for (int i = 0; i < down.length; i++) {
        for (int j = 0; j < down.length; j++) {
            down[i][j] = arr[i][j]; // storing the array for
down shift
        }
    }
    if (blankRowIndex != arr.length - 1) { // checks if the
down shift is possible and doesn't go out of bounds
        int temp = down[blankRowIndex +
1][blankColIndex];
        down[blankRowIndex + 1][blankColIndex] =
down[blankRowIndex][blankColIndex];
        down[blankRowIndex][blankColIndex] = temp;
    }

    if (mismatch(down) + level <= cost) { // checking if
the cost is lower
        isChoosen = "Shifting Down";
        cost = mismatch(down) + level;
        for (int i = 0; i < left.length; i++) {
            for (int j = 0; j < left.length; j++) {
                temp_array[i][j] = down[i][j];
            }
        }
    }
    return temp_array;
```

```
   }
}

public class puzzleSolver {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      branchBounds obj = new branchBounds();
      System.out.println("--------------15 puzzle solve----------");
      System.out.println("Input Matrix: ");
      System.out.print("\nEnter the size of the matrix: ");
      int size = sc.nextInt();
      int[][] table = new int[size][size];
      System.out.print("\nEnter the elements of the matrix: ");
      for (int i = 0; i < size; i++) {
         for (int j = 0; j < size; j++) {
            table[i][j] = sc.nextInt();
         }
      }

      System.out.println("The Length of the puzzle is: " + table.length);
      if (obj.isSolvable(table, table[0].length)) {
         System.out.println("\nPuzzle is solvable");
         obj.solve(table);// solving the matrix

      } else {
         System.out.println("\n Puzzle is not solvable");
      }
      sc.close();

   }
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

SCREENSHOT:

Not Solvable

```
--------------15 puzzle solve----------------
Input Matrix:

Enter the size of the matrix: 4
The Length of the puzzle is: 4
----------------------------
|   3  |   9  |   1  |  15  |
----------------------------
|  14  |  11  |   4  |   6  |
----------------------------
|  13  |   0  |  10  |  12  |
----------------------------
|   2  |   7  |   8  |   5  |
----------------------------

----------------------
X Mark is at -> 3, 2
----------------------
Total inversions: 56

 Puzzle is not solvable
```

Solvable

```
Input Matrix:

Enter the size of the matrix: 4

Enter the elements of the matrix: 1 2 3 0 5 6 7 4 9 10 11 8 13 14 15 12
The Length of the puzzle is: 4
----------------------------
|   1  |   2  |   3  |   0  |
----------------------------
|   5  |   6  |   7  |   4  |
----------------------------
|   9  |  10  |  11  |   8  |
----------------------------
|  13  |  14  |  15  |  12  |
----------------------------

----------------------
X Mark is at -> 1, 4
----------------------
Total inversions: 9

Puzzle is solvable
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

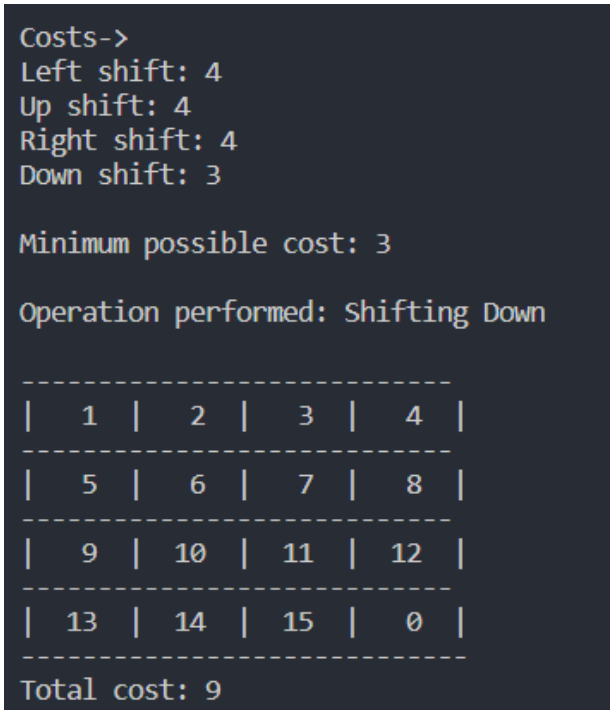
```
Costs->
Left shift: 5
Up shift: 4
Right shift: 4
Down shift: 3

Minimum possible cost: 3

Operation performed: Shifting Down

-----------------------------
|   1  |   2  |   3  |   4  |
-----------------------------
|   5  |   6  |   7  |   0  |
-----------------------------
|   9  |  10  |  11  |   8  |
-----------------------------
|  13  |  14  |  15  |  12  |
-----------------------------
```

```
Costs->
Left shift: 4
Up shift: 4
Right shift: 4
Down shift: 3

Minimum possible cost: 3

Operation performed: Shifting Down

-----------------------------
|   1  |   2  |   3  |   4  |
-----------------------------
|   5  |   6  |   7  |   8  |
-----------------------------
|   9  |  10  |  11  |   0  |
-----------------------------
|  13  |  14  |  15  |  12  |
-----------------------------
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

	
TIME COMPLEXITY	The Time complexity of the branch and Bound algorithm is given below: **Time Complexity: O(n^2)** The Space Complexity of the branch and Bound Algorithm is given below: **Space Complexity: O(n)**

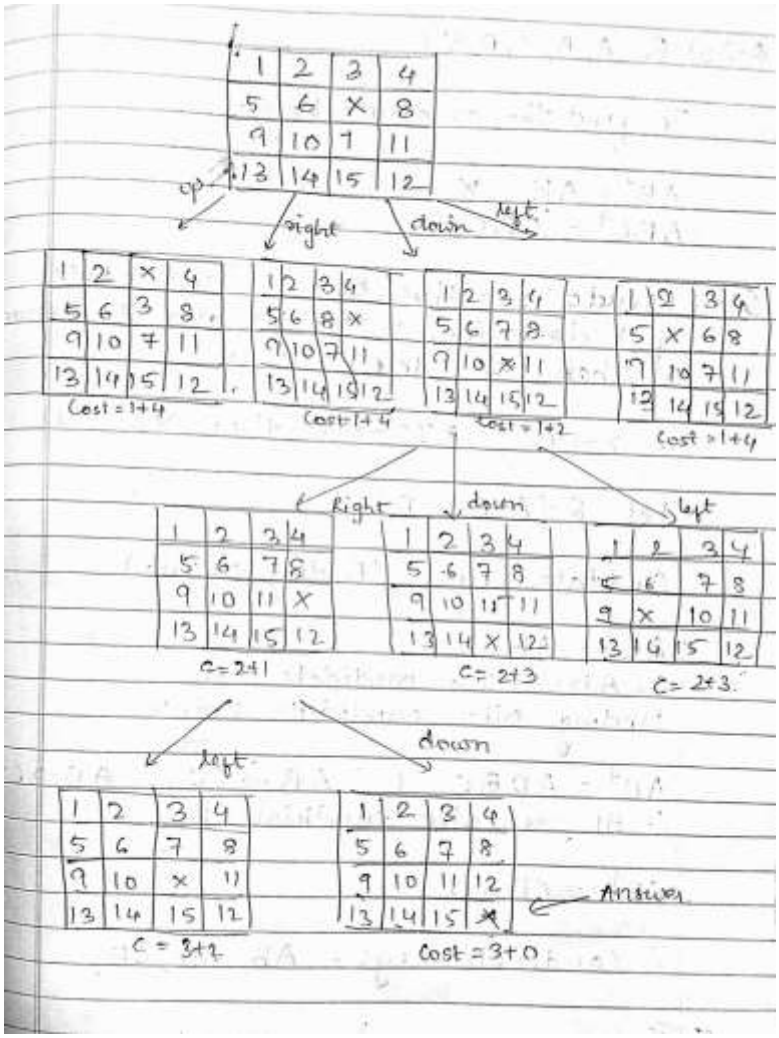
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

OUTPUT:



CONCLUSION: Things learnt during the procedural programming of the problem of branch and bound

- Learnt one of the most popular algorithms used in the optimization problem is the branch and bound algorithm.
- Learnt about some advantages and disadvantages of the branch and bound algorithm
- Also noted how and when a branch and bound algorithm would be the right choice for a user to use.
- Used branch and bound based algorithm for solving the 15 puzzle problem.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	9		

AIM:	To implement approximation algorithms
THEORY:	**What is the Approximation algorithm?** An approximation algorithm is a way of dealing with NP-completeness for an optimization problem. This technique does not guarantee the best solution. The goal of the approximation algorithm is to come close as much as possible to the optimal solution in polynomial time. Such algorithms are called approximation algorithms or heuristic algorithms. **Features of Approximation Algorithm :** Here, we will discuss the features of the Approximation Algorithm as follows. • An approximation algorithm guarantees to run in polynomial time though it does not guarantee the most effective solution. • An approximation algorithm guarantees to seek out high accuracy and top quality solution(say within 1% of optimum) • Approximation algorithms are used to get an answer near the (optimal) solution of an optimization problem in polynomial time **Performance Ratios** Suppose we work on an optimization problem where every solution carries a cost. An Approximate Algorithm returns a legal solution, but the cost of that legal solution may not be optimal.

For Example, suppose we are considering for a minimum size vertex-cover (VC). An approximate algorithm returns a VC for us, but the size (cost) may not be minimized.

Another Example is we are considering for a maximum size Independent set (IS). An approximate Algorithm returns an IS for us, but the size (cost) may not be maximum. Let C be the cost of the solution returned by an approximate algorithm, and C* is the cost of the optimal solution.

We say the approximate algorithm has an approximate ratio P (n) for an input size n, where

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq P(n)$$

Intuitively, the approximation ratio measures how bad the approximate solution is distinguished with the optimal solution. A large (small) approximation ratio measures the solution is much worse than (more or less the same as) an optimal solution.

Applications of Approximation algorithm :

1. The Vertex Cover Problem

In the vertex cover problem, the optimization problem is to find the vertex cover with fewest vertices, and the approximation problem is to find the vertex cover with few vertices.

2. Travelling Salesman Problem

In the traveling salesperson problem, the optimization problem is to find the shortest cycle, and the approximation problem is to find a short cycle.

3. The Set Covering Problem

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

This is an optimization problem that models many problems that require resources to be allocated. Here, a logarithmic approximation ratio is used.

4. The Subset Sum Problem

In the Subset sum problem, the optimization problem is to find a subset of {x1,×2,×3…xn} whose sum is as large as possible but not larger than target value t.

Set Covering Problem

```
U = {1,2,3,4,5}
S = {S1,S2,S3}

S1 = {4,1,3},    Cost(S1) = 5
S2 = {2,5},      Cost(S2) = 10
S3 = {1,4,3,2},  Cost(S3) = 3

Output: Minimum cost of set cover is 13 and
        set cover is {S2, S3}

There are two possible set covers {S1, S2} with cost 15
and {S2, S3} with cost 13.
```

PSEUDOCODE:

Algorithm

1. $C \leftarrow 0$
2. While $C \neq U$ do

   Find the set whose cost effectiveness is smallest, say S

   Let $\alpha = \frac{c(S)}{|S - C|}$

   For each $e \in S\text{-}C$, set price(e) = α

   $C \leftarrow C \cup S$
3. Output picked sets

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	
CODE:	import java.util.HashSet; import java.util.Scanner; import java.util.Set; public class set_covering { *// calculate minimum cost of set cover* public static void minCost(Set<Integer> Univ, Set<Integer> S[], int cost[], int m) { *// Set I is used to get minimum cost* Set<Integer> I = new HashSet<Integer>(); int minCost = 0; System.out.print("Best possible Solution sets\n"); *// Loop continues till the time I contains all the elements of U* while (!I.equals(Univ)) { *// calculate the min cost* double min = Double.MAX_VALUE; *// loop to find the minimum cost* int index = -1; double effCost[] = new double[m]; for (int i = 0; i < m; i++) { *// diff is difference between S[i] and I* Set<Integer> diff = new HashSet<Integer>(S[i]); diff.removeAll(I); *// if size of diff != 0* if (diff.size() != 0) { effCost[i] = cost[i] / diff.size(); } *// if size of diff = 0* else { effCost[i] = Double.MAX_VALUE; } *// change min if effCost[i] < min* if (min > effCost[i]) {

```
                min = effCost[i];
                index = i;
            }
        }
        // Printing the S[i]
        System.out.print("S[" + (index + 1) + "] ");
        // Used to get union of I and S[index]
        Set<Integer> union = new HashSet<Integer>(I);
        union.addAll(S[index]);
        // I = union of I and S[index]
        I = union;
        // Min cost
        minCost = minCost + cost[index];
    }
    System.out.println(" Minimum cost: " + minCost);
  }

  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    // no of elements in U

    System.out.println("------------------SET COVERING
PROBLEM------------------\n");
    System.out.print("Enter the elements in Universal Set
: ");
    int n = sc.nextInt();
    // for storing the values in Universal set
    System.out.print("Enter the elements -> ");
    Set<Integer> Univ = new HashSet<Integer>();
    for (int i = 0; i < n; i++) {
      int temp = sc.nextInt();
      Univ.add(temp);
    }
    // no of sets in S
    System.out.print("Enter the number of sets : ");
    int m = sc.nextInt();
    // for storing the values in sets
    Set<Integer>[] S = new HashSet[m];
    int cost[] = new int[m];
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        for (int i = 0; i < m; i++) {
            S[i] = new HashSet<Integer>();
            System.out.println("\nSet [" + (i + 1) + "] Details:
");
            // no of elements in S[i]
            System.out.print("Enter the elements : ");
            String input = sc.next();
            String[] elements = input.split(" ");
            for (int j = 0; j < elements.length; j++) {
                S[i].add(Integer.parseInt(elements[j]));
            }
            // Cost of S[i]
            System.out.print("Enter the cost(S[" + (i + 1) + "])
: ");
            cost[i] = sc.nextInt();
        }
        // method to get min cost
        minCost(Univ, S, cost, m);
        sc.close();
    }
}
```

OUTPUT:

```
------------ Subset Covering Prob ------------

Enter the elements in Universal Set : 5
Enter the elements -> 1 2 3 4 5
Enter the number of sets : 4

--------S[1]--------

Enter the elements of S[1]: 2
Enter the elements : 5 4
Enter the cost(S[1]) : 3

--------S[2]--------

Enter the elements of S[2]: 3
Enter the elements : 1 2 3
Enter the cost(S[2]) : 5
```

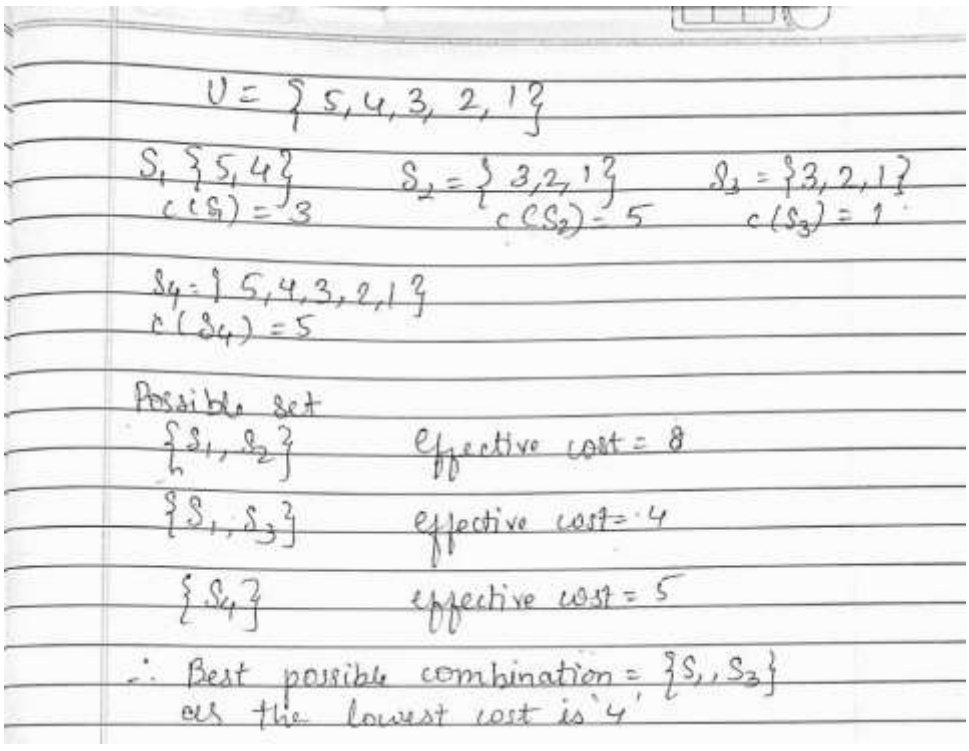
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
--------S[3]--------

Enter the elements of S[3]: 3
Enter the elements : 1 2 3
Enter the cost(S[3]) : 1

--------S[4]--------

Enter the elements of S[4]: 5
Enter the elements : 1 2 3 4 5
Enter the cost(S[4]) : 5
Best possible Solution sets is:
->S[3] S[1]  Minimum cost: 4
```

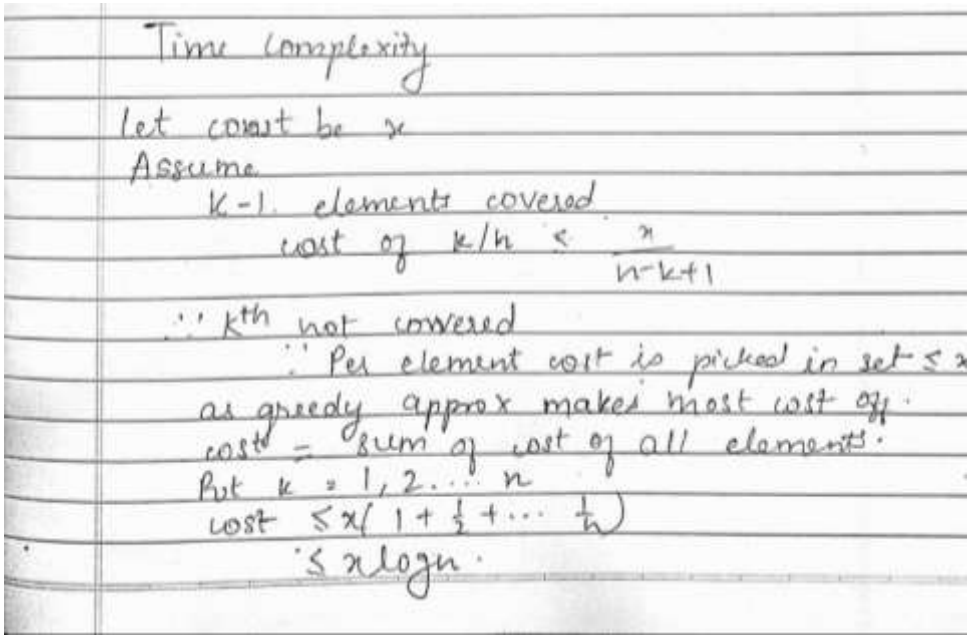


Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

TIME COMPLEXITY	 **TIME & SPACE COMPLEXITY** The time complexity for approximation problem using greedy approximation approach is found out to be O(log n).

CONCLUSION: After the procedural programming through this experiment:

- Learnt about NP complete problem and how approximation algorithm is used to solve it.
- Implemented Set covering problem where used sets for storing the subsets and calculated the minimum cost for covering the sets.
- Derived the Time Complexity for the Set Covering Problem using Approximation algorithm and found it to be O(log n).
- Further for more explanation, I solved the Subset Covering Problem and for presented its working and commented on the time complexity which was derived and justified the derivation.

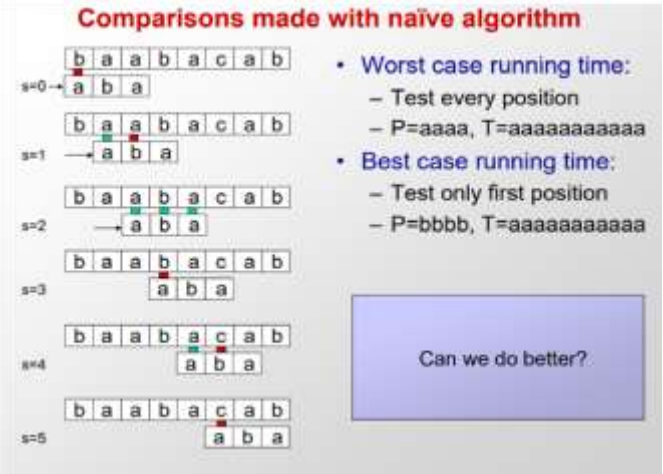
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	10		

AIM:	Given a universe U of n elements, a collection of subsets of U say S = {S1, S2..., Sm} where every subset Si has an associated cost. Find a minimum cost subcollection of S that covers all elements of U.
THEORY:	NAÏVE STRING MATCHING The naïve approach tests all the possible placement of Pattern P [1.....m] relative to text T [1.....n]. We try shift s = 0, 1.....n-m, successively and for each shift s. Compare T [s+1.....s+m] to P [1.....m]. The naïve algorithm finds all valid shifts using a loop that checks the condition P [1.....m] = T [s+1.....s+m] for each of the n - m +1 possible value of s. **NAIVE-STRING-MATCHER (T, P)** 1. n ← length [T] 2. m ← length [P] 3. for s ← 0 to n -m 4. do if P [1.....m] = T [s + 1....s + m] 5. then print "Pattern occurs with shift" s 

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

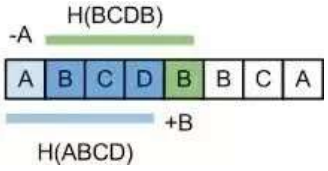
Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

	CALCULATION OF VALUE OF H IN TC O(LOG(M)). HINT MODULAR EXPONENTIATION Modular exponentiation is exponentiation performed over a modulus. Modular exponentiation is the remainder when an integer b (the base) is raised to the power e (the exponent), and divided by a positive integer m (the modulus); i.e., c = be mod m. It is O(log(m)) because the value is calculated as the sum of geometric progression depending on the value of d and m along with using the modulus with q. **ROLLING HASH:** Consider this example. Let's say you have a string "abcdeacdoe" & you want to find the pattern "bcd" in this string. Now with the naive way, you will try to compare each character of pattern ("bcd") with the string characters ("abcdbbca") forming three-character strings. This might be really inefficient especially as the input gets larger. If you use rolling hash, you can do this intelligently. First you calculate the hash of first three letter substring (abc) in the string. To keep matters simple, let's say we use base 7 for this calculation (in the actual scenario we should mod it with a prime to avoid overflow): **substring1** = a*7^0+ b*7^1+c*7^2=97*1+98*7+99*49=5634 **pattern** = b*7^0+ c*7^1+d*7^2=98*1+99*7+100*49=5691 So you compare two hash values and since they are different, you move forward. Now you reach to second substring "bcd" in string. Here is where the fun begins. We can calculate the hash of this string without rehashing everything. As you can see, the window has moved forward only by dropping one character and adding another: a<-bc<-d Below diagrams explains it better (though it's working on a four-letter string rather than three):

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



** credits: quora*

From hash value we drop the first character value, divide the left out value with the base number and add the new char to the rightmost position with appropriate power of base: Here are the steps:
drop a => (a*7^0+ b*7^1+c*7^2) - a*7^0
divide everything by 7 => (b*7^1+c*7^2)/7 => b*7^0 + c*7^1
add d => b*7^0 + c*7^1 + d*7^2 [the power of base for d is (pattern-length-1)]
Thus new hash for "bcd" in input string would be => (5634 - 97)/7+ 100*49 = 5691 which matches pattern hash value. Now we can go ahead and compare the strings to verify if they are in fact same.
Visualize doing this for a large string which comprising of lets say, 30 lines. You will be able to get new hash value in constant time without much effort by just dropping and adding new character(s).
In real use cases, to avoid overflow because of large power, we will have to do modulo with large prime. This technique is often used for multi-pattern string search and forms the core of algorithms like the Karp-Rabin algorithm. This algorithm is an excellent choice to detect plagiarism.

WHY ARE PRIME NUMBERS USED FOR CONSTRUCTING HASH FUNCTIONS?

The reason why prime numbers are used is to minimize collisions when the data exhibits some particular patterns.
First things first: If the data is random then there's no need for a prime number, you can do a mod operation against any number and you will have the same number of collisions for each possible value of the modulus.
But when data is not random then strange things happen. For example, consider numeric data that is always a multiple of 10.

If we use mod 4 we find:
10 mod 4 = 2
20 mod 4 = 0
30 mod 4 = 2
40 mod 4 = 0 50 mod 4 = 2
So from the 3 possible values of the modulus (0, 1, 2, 3), only 0 and 2 will have collisions, that is bad.
If we use a prime number like 7:
10 mod 7 = 3
20 mod 7 = 6
30 mod 7 = 2
40 mod 7 = 4
50 mod 7 = 1
We also note that 5 is not a good choice but 5 is prime the reason is that all our keys are a multiple of 5. This means we have to choose a prime number that doesn't divide our keys, choosing a large prime number is usually enough.
So erring on the side of being repetitive the reason prime numbers are used is to neutralize the effect of patterns in the keys in the distribution of collisions of a hash function.

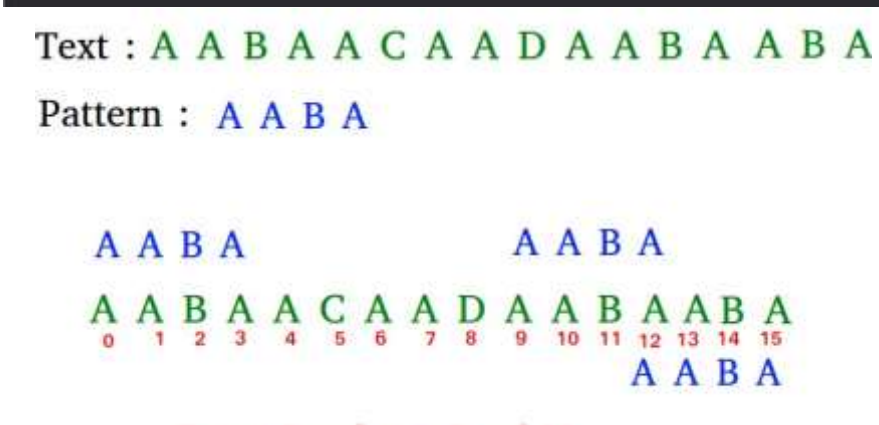
THE RABIN-KARP ALGORITHM
Like the **Naive Algorithm**, the Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, the Rabin Karp algorithm **matches the hash value** of the pattern with the hash value of the current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for the following strings.

1. Pattern itself.
2. All the substrings of the text of length m.

Since we need to efficiently calculate hash values for all the substrings of size m of text, we must have a hash function that has the following property.
Hash at the next shift must be efficiently computable from the current hash value and next character in text or we can say hash(txt[s+1 .. s+m]) must be efficiently computable from hash(txt[s .. s+m-1]) and txt[s+m] i.e., hash(txt[s+1

.. s+m])= rehash(txt[s+m], hash(txt[s .. s+m-1])) and rehash must be O(1) operation.
The hash function suggested by Rabin and Karp calculates an integer value. The integer value for a string is the numeric value of a string.

```
Input:  txt[] = "THIS IS A TEST TEXT"
        pat[] = "TEST"
Output: Pattern found at index 10

Input:  txt[] =  "AABAACAADAABAABA"
        pat[] =  "AABA"
Output: Pattern found at index 0
        Pattern found at index 9
        Pattern found at index 12
```



**credits: documentation*

Limitations of the Rabin-Karp Algorithm

Spurious Hit

When the hash value of the pattern matches with the hash value of a window of the text but the window is not the actual pattern then it is called a spurious hit.
Spurious hit increases the time complexity of the algorithm.
In order to minimize spurious hits, we use modulus. It greatly reduces the spurious hit.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

PSEUDOCODE:	n = t.length m = p.length h = dm-1 mod q p = 0 t0 = 0 for i = 1 to m p = (dp + p[i]) mod q t0 = (dt0 + t[i]) mod q for s = 0 to n - m if p = ts if p[1.....m] = t[s + 1..... s + m] print "pattern found at position" s If s < n-m ts + 1 = (d (ts - t[s + 1]h) + t[s + m + 1]) mod q
EXPERIMENT 10	
CODE:	import java.util.ArrayList; import java.util.Scanner; public class rabinKarp { static ArrayList<Integer> locations = new ArrayList<Integer>(); *// d is the number of characters in the input alphabet* *public final static int* static void search(String pat, String txt, int q) { int d = 256; int M = pat.length(); int N = txt.length(); int i, j; int p = 0; *// hash value for pattern* int t = 0; *// hash value for txt* int h = 1; *// The value of h would be "pow(d, M-1)%q"* for (i = 0; i < M - 1; i++) h = (h * d) % q; *// Calculate the hash value of pattern and first*

```
        // window of text
        for (i = 0; i < M; i++) {
            p = (d * p + pat.charAt(i)) % q;
            t = (d * t + txt.charAt(i)) % q;
        }
        // Slide the pattern over text one by one
        for (i = 0; i <= N - M; i++) {
            // Check the hash values of current window of text
            // and pattern. If the hash values match then only
            // check for characters on by one if (p == t) {
            /* Check for characters one by one */
            for (j = 0; j < M; j++) {
                if (txt.charAt(i + j) != pat.charAt(j))
                    break;
            }
            // if p == t and pat[0...M-1] = txt[i, i+1,...i+M-1]
            if (j == M) {
                locations.add(i);
                System.out.println("Pattern found at index " +
i);
            }
        }
        // Calculate hash value for next window of text:
Remove
        // leading digit, add trailing digit
        if (i < N - M) {
            t = (d * (t - txt.charAt(i) * h) + txt.charAt(i + M))
% q;
            // We might get negative value of t, converting it
            // to positive
            if (t < 0)
                t = (t + q);
        }
    }

    /* Driver program to test above function */
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter the text: ");
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        String txt = sc.next();
        System.out.print("\nEnter the pattern: ");
        String pat = sc.next();
        int q = 101; // A prime number
        search(pat, txt, q);
        System.out.print("\nPattern found at following
locations: \n");
        for (int i = 0; i < txt.length(); i++) {
            System.out.print(txt.charAt(i));
        }
        System.out.print("\n");
        for (int i = 0; i < txt.length() && locations.size() != 0;
i++) {
            if (i == locations.get(0)) {
                System.out.print("^");
                locations.remove(0);
            } else
                System.out.print(" ");
        }
        sc.close();
    }
}
```

OUTPUT:

```
Enter the text: aabadjwdaaab

Enter the pattern: aab
Pattern found at index 0
Pattern found at index 9

Pattern found at following locations:
aabadjwdaaab
^         ^
```

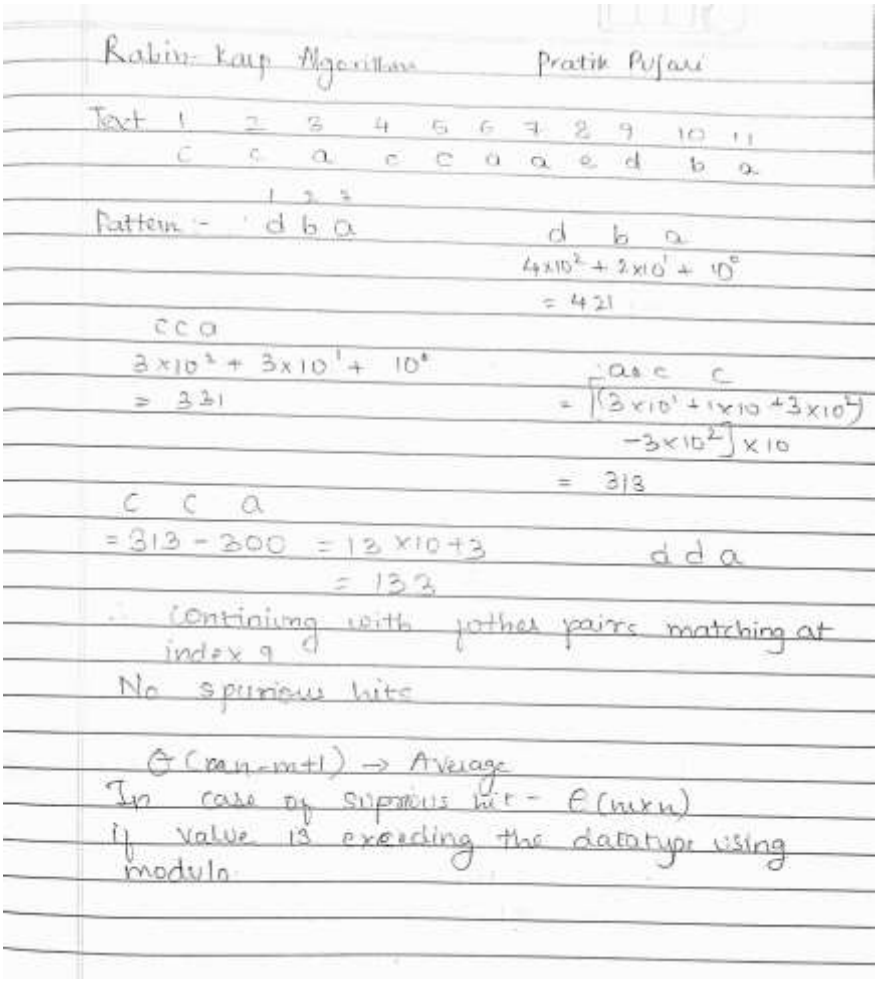
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
Enter the text: helloworldworlhelloworld

Enter the pattern: ellow
Pattern found at index 1
Pattern found at index 15

Pattern found at following locations:
helloworldworlhelloworld
 ^              ^
```



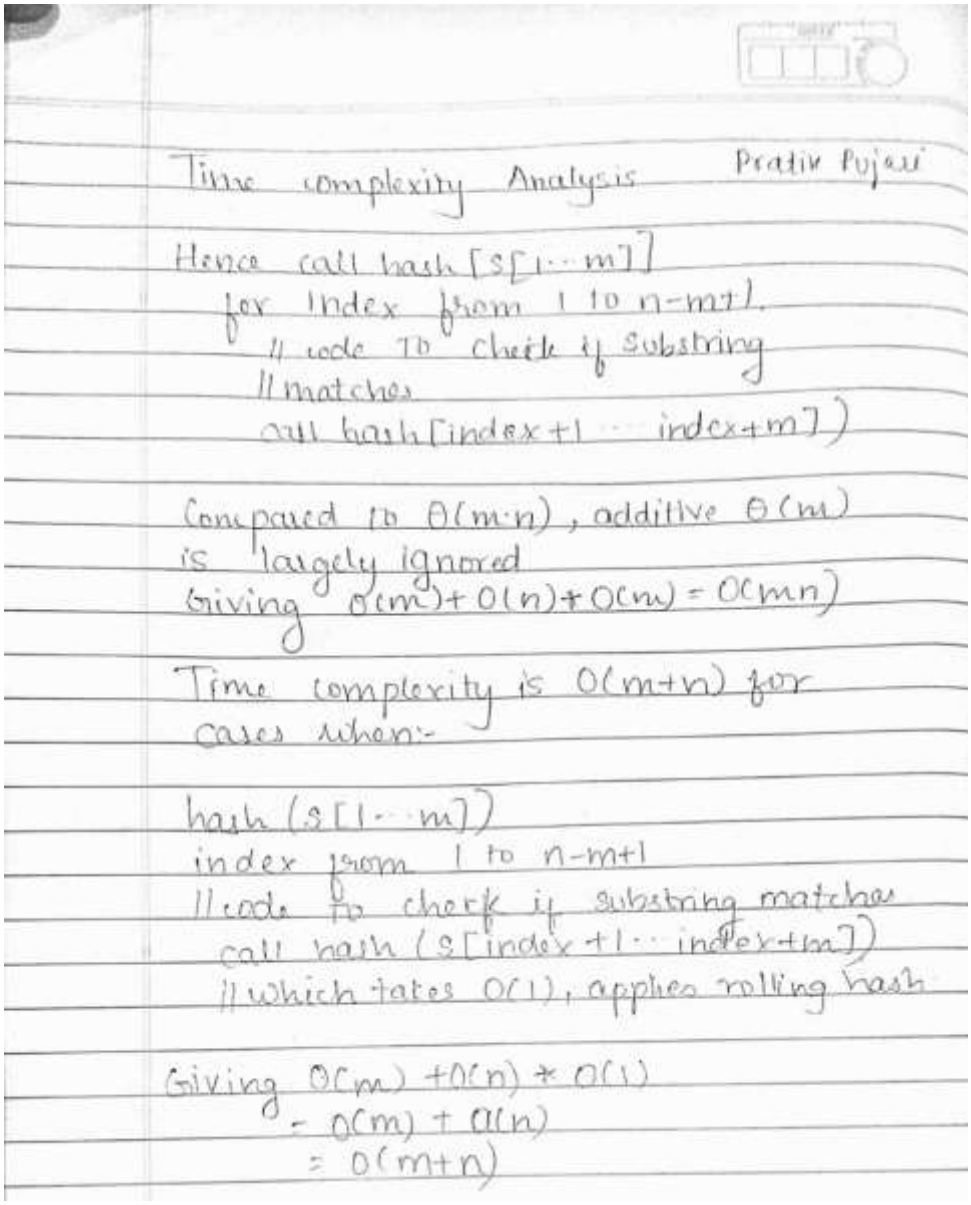
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

TIME COMPLEXITY:



Assume the text is length n and the length of the word is m. The **best- and average-case** running time of Rabin-Karp is **O(m+n)** because the rolling hash step takes O(n) time and once the algorithm finds a potential match, it must verify each letter to make sure that the match is true

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

	The **worst-case** running time of Rabin-Karp is **O(nm)**. This would occur with an extremely awful hash function that resulted in a false positive at each step.

CONCLUSION: Things leant during the procedural programming of the problem

- Learnt the concept related to most of the String matching algorithms.
- Implemented Rabin Karp algorithm where the hash value of the text and the pattern is calculated and found out all the answers to additional questions assigned.
- Learnt how to find out the best and worst case time complexity.
- Also learnt how to solve Rabin Karp problems on paper.