Experiment **#8**		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

Experiment Title: To implement programs on problem solving using Greedy Approach. – Scenario2.

Aim/Objective: To understand the concept and implementation of Basic programs on Greedy Approach based Problems.

Description: The students will understand and able to implement programs on Greedy Approach based Problems.

Pre-Requisites:

Knowledge: Greedy method and its related problems in C/ java/Python.

Tools: Code Blocks/Eclipse IDE.

Pre-Lab:

1. A group of children are seated in a row, each with a different performance score. Tom wants to distribute candies in such a way that: Every child must receive at least 1 candy. Children with higher performance scores than their neighbors should receive more candies. The total number of candies should be minimized. Your task is to determine the total number of candies Tom should distribute.

Input Format:

- The first line contains an integer n (the number of children).
- The next n lines contain an integer representing the performance score of each child.

Output :

The total minimum number of candies that need to be distributed.

Sample Input:

4

1 2 3 4

Sample Output:

10

- **Procedure/Program**:

```
#include <stdio.h>

int minCandies(int scores[], int n) {
   int candies[n];
   for (int i = 0; i < n; i++) {
      candies[i] = 1;
   }

   for (int i = 1; i < n; i++) {
      if (scores[i] > scores[i - 1]) {
         candies[i] = candies[i - 1] + 1;
      }
   }

   for (int i = n - 2; i >= 0; i--) {
      if (scores[i] > scores[i + 1]) {
         candies[i] = candies[i] > candies[i + 1] + 1 ? candies[i] : candies[i + 1] + 1;
      }
   }

   int totalCandies = 0;
   for (int i = 0; i < n; i++) {
      totalCandies += candies[i];
   }

   return totalCandies;
}

int main() {
   int n = 4;
   int scores[] = {1, 2, 3, 4};
   printf("%d\n", minCandies(scores, n));
   return 0;
}
```

- **Data and Results:**

Data

The input consists of an integer `n` followed by children's performance scores.

Result

Total minimum number of candies required for distribution is 10.

- **Analysis and Inferences:**

Analysis

The algorithm distributes candies based on relative performance scores of children.

Inferences

Greedy approach ensures fairness while minimizing total candy distribution.

2. Given A shop sells items, and a group of friends wants to buy all of them. Each item's price increases based on how many items that person has already purchased. Find the minimum cost to buy all items by distributing purchases among friends.

Input:

6 4

7 1 3 4 5 6

Output :

38

Explanation:

Assign the most expensive items first to distribute the increasing cost among friends.

Procedure/Program:

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

int compare(const void *a, const void *b) {
   return *(int *)b - *(int *)a;
}

int minCost(int items[], int n, int friends) {
   qsort(items, n, sizeof(int), compare);

   int cost = 0, count[friends];
   for (int i = 0; i < friends; i++) count[i] = 0;

   for (int i = 0; i < n; i++)
      cost += (count[i % friends] + 1) * items[i], count[i % friends]++;
```

```
    return cost;
}

int main() {
    int n = 6, friends = 4;
    int items[] = {7, 1, 3, 4, 5, 6};

    printf("%d\n", minCost(items, n, friends));
    return 0;
}
```

- **Data and Results:**

Data

Shop sells items, prices increase with each purchase, distribute wisely.

Result

Minimum cost to buy all items is calculated efficiently.

- **Analysis and Inferences:**

Analysis

Sorting items in descending order minimizes cost by distributing purchases.

Inferences

Optimal distribution among friends reduces overall expense significantly.

In-Lab:

1. The absolute difference is the positive difference between two values a and b, is written |a-b| or |b-a| and they are equal. If a=3 and b=2, |3-2|=|2-3|=1. Given an array of integers, find the minimum absolute difference between any two elements in the array.

Input Format

Arr= [-2, 2, 4]

There are 3 pairs of numbers: [-2, 2], [-2, 4] and [2, 4]. The absolute differences for these pairs are| (-2)-2|=4, | (-2)-4 |=6and |2-4|=2. The minimum absolute difference is 2.

Function Description:

Complete the minimum Absolute Difference function in the editor below. It should return an integer that represents the minimum absolute difference between any pair of elements.

Minimum Absolute Difference has the following parameter(s):

Int arr[n]: an array of integers

Returns int: the minimum absolute difference found

Input Format:

- The first line contains a single integer n, the size of arr.
- The second line contains n space-separated integers, arr[i].

Sample Input

3

3 -7 0

Sample Output

3

- **Procedure/Program**:

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

int minimumAbsoluteDifference(int arr[], int n) {
   int minDiff = abs(arr[0] - arr[1]);

   for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
         int diff = abs(arr[i] - arr[j]);
         if (diff < minDiff) {
            minDiff = diff;
         }
      }
   }

   return minDiff;
}
int main() {
   int arr[] = {3, -7, 0};
   int n = sizeof(arr) / sizeof(arr[0]);
   int result = minimumAbsoluteDifference(arr, n);
   printf("%d\n", result);
   return 0;
}
```

- **Data and Results:**

Data

Given an array, find the minimum absolute difference efficiently.

Result

The minimum absolute difference between any two elements is determined.

- **Analysis and Inferences:**

Analysis

Sorting helps optimize the search for the minimum absolute difference.

Inferences

Pairwise comparison without sorting increases time complexity significantly.

2. You have a list of movies, each with a certain "energy cost" associated with watching it. After watching a movie, the effort required for the next movie increases exponentially. Specifically, if you have already watched i movies, the next movie will require 2i ×energy cost energy. Your task is to determine the minimum total energy required to watch all the movies, given that you can watch them in any order.

Input Format :

- An integer n, representing the number of movies.
- An array energy [] of size n, where energy[i] represents the energy cost of watching the ith movie.

Output Format:

- A single integer representing the minimum total energy required to watch all movies.

Sample Input:

3
[3, 2, 5]

Sample Output:

17

- **Procedure/Program**:

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

int compare(const void *a, const void *b) {
   return (*(int*)a - *(int*)b);
}

int min_total_energy(int n, int energy[]) {
   qsort(energy, n, sizeof(int), compare);
   int total_energy = 0, factor = 1;

   for (int i = 0; i < n; i++) {
      total_energy += factor * energy[i];
      factor *= 2;
   }
```

```
    return total_energy;
}

int main() {
    int n = 3;
    int energy[] = {3, 2, 5};
    printf("%d\n", min_total_energy(n, energy));
    return 0;
}
```

- **Data and Results:**

Data

Movies have energy costs, and watching order impacts total energy.

Result

Optimal order minimizes exponential energy growth, reducing total consumption.

- **Analysis and Inferences:**

Analysis

Sorting movies in ascending order ensures minimal exponential energy increase.

Inferences

Efficient sequencing significantly reduces total effort for movie watching.

Post-Lab:

Given an array of ages of size n, where each element represents the age of a person, the task is to group people into different age groups. Each group should contain people whose ages have a difference of at most 2 years. Return the groups formed.

Input Format:

An integer array ages [] of size N.

Output:

A list of groups, where each group is a list containing people within an age difference of at most 2 years.

Sample Input:

ages = [25, 30, 35, 32, 28, 23, 40, 45, 22].

Input Format:

[[25, 23], [30, 32, 28], [35, 40], [45], [22]].

- **Procedure/Program**:

```
#include <stdio.h>

#include <stdlib.h>


int compare(const void *a, const void *b) {

   return (*(int *)a - *(int *)b);

}


void groupAges(int ages[], int n) {

   qsort(ages, n, sizeof(int), compare);
```

```
    int i = 0;
    while (i < n) {
        printf("[");
        int start = i;
        printf("%d", ages[i]);
        i++;
        while (i < n && ages[i] - ages[start] <= 2) {
            printf(", %d", ages[i]);
            i++;
        }
        printf("] ");
    }
}
int main() {
    int ages[] = {25, 30, 35, 32, 28, 23, 40, 45, 22};
    int n = sizeof(ages) / sizeof(ages[0]);
    groupAges(ages, n);
    return 0;
}
```

- **Data and Results:**

Data

The given ages are grouped based on a two-year difference.

Result

People are categorized into groups maintaining a maximum age gap.

- **Analysis and Inferences:**

Analysis

Sorting helps in efficiently forming age groups with minimal operations.

Inferences

Age-based grouping simplifies understanding demographic distributions and trends.

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the Traveling Salesman Problem (TSP)?

Visit all cities once, return, minimize cost.

2. Compare Greedy Algorithms and Divide and Conquer.

Greedy picks locally best; Divide & Conquer splits, solves, merges.

3. How does the greedy strategy work in job scheduling with deadlines?

Sort by profit, schedule latest before deadline.

4. Why is TSP classified as an NP-hard problem?

No polynomial-time solution, requires checking all routes.

5. What is the time complexity of Dijkstra's algorithm?

O((V + E) log V) with a priority queue.

Evaluator Remark (if Any):	**Marks Secured ___out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.