**DAA PRACTICAL NO : 01**

|  |  |
| --- | --- |
| **Name :** | **Kunal Nayak** |
| **Roll No :** | **A2-B4-53** |
| **Date :** | **22-07-25** |

**Task A**

**Code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**float getRandomFloat(float min, float max) {**

**return min + ((float) rand() / RAND\_MAX) \* (max - min);**

**}**

**void generateData(float temp[], float pressure[], int n) {**

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

**temp[i] = getRandomFloat(-20, 50);**

**pressure[i] = getRandomFloat(950, 1050);**

**}**

**}**

**int findMinTemperature(float temp[], int n) {**

**int minIndex = 0;**

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

**if (temp[i] < temp[minIndex]) {**

**minIndex = i;**

**}**

**}**

**return minIndex;**

**}**

**int findMaxPressure(float pressure[], int n) {**

**int maxIndex = 0;**

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

**if (pressure[i] > pressure[maxIndex]) {**

**maxIndex = i;**

**}**

**}**

**return maxIndex;**

**}**

**int main() {**

**srand(time(0)); // Add this line to seed randomness**

**int n = 100;**

**float temp[n], pressure[n];**

**generateData(temp, pressure, n);**

**clock\_t start, end;**

**double duration;**

**start = clock();**

**int minTempIndex = findMinTemperature(temp, n);**

**end = clock();**

**duration = (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("Minimum Temperature: %.2f °C at index %d and Time: %lf seconds\n",**

**temp[minTempIndex], minTempIndex, duration);**

**start = clock();**

**int maxPressureIndex = findMaxPressure(pressure, n);**

**end = clock();**

**duration = (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("Maximum Pressure: %.2f hPa at index %d and Time: %lf seconds\n",**

**pressure[maxPressureIndex], maxPressureIndex, duration);**

**return 0;**

**}**

**Task B**

**Code :**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**float getRandomFloat(float min, float max) {**

**return min + ((float) rand() / RAND\_MAX) \* (max - min);**

**}**

**void generateData(float temp[], float pressure[], int n) {**

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

**temp[i] = getRandomFloat(-20, 50); // Temperature range**

**pressure[i] = getRandomFloat(950, 1050); // Pressure range**

**}**

**}**

**int naiveFindMin(float arr[], int n) {**

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

**int isMin = 1;**

**for (int j = 0; j < n; j++) {**

**if (arr[j] < arr[i]) {**

**isMin = 0;**

**break;**

**}**

**}**

**if (isMin)**

**return i;**

**}**

**return -1;**

**}**

**int naiveFindMax(float arr[], int n) {**

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

**int isMax = 1;**

**for (int j = 0; j < n; j++) {**

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

**isMax = 0;**

**break;**

**}**

**}**

**if (isMax)**

**return i;**

**}**

**return -1;**

**}**

**int main() {**

**srand(time(0)); // ✅ Important to get new random data every run**

**int n = 100;**

**float temp[n], pressure[n];**

**generateData(temp, pressure, n);**

**clock\_t start, end;**

**double duration;**

**// Min Temperature**

**start = clock();**

**int minTempIndex = naiveFindMin(temp, n);**

**end = clock();**

**duration = (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("Minimum Temperature: %.2f °C at index %d and Time: %lf seconds\n",**

**temp[minTempIndex], minTempIndex, duration);**

**// Max Pressure**

**start = clock();**

**int maxPressureIndex = naiveFindMax(pressure, n);**

**end = clock();**

**duration = (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("Maximum Pressure: %.2f hPa at index %d and Time: %lf seconds\n",**

**pressure[maxPressureIndex], maxPressureIndex, duration);**

**return 0;**

**}**

**Output :**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task** | **Loop Type** | **Time**  **Complexity** | **Parameters** | **n = 10^2** | **n = 10^4** | **n = 10^6** |
| **Task-A** | **Linear** | **O(n)** | **Temperature**  **Pressure** |  |  |  |
| **Task-B** | **Quadratic** | **O(n^2)** | **Temperature**  **Pressure** |  |  |  |

**Task C**

**Code**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**// Generates random float between min and max**

**float getRandomFloat(float min, float max) {**

**return min + ((float) rand() / RAND\_MAX) \* (max - min);**

**}**

**void generateRandomSortedTemps(float arr[], int n) {**

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

**arr[i] = getRandomFloat(20.0f, 50.0f); // random between 20 and 50**

**}**

**// Simple bubble sort (for small n) – replace with qsort if needed**

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

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

**if (arr[j] > arr[j + 1]) {**

**float temp = arr[j];**

**arr[j] = arr[j + 1];**

**arr[j + 1] = temp;**

**}**

**}**

**}**

**}**

**int linearSearch(float arr[], int n) {**

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

**if (arr[i] >= 30.0)**

**return i;**

**}**

**return -1;**

**}**

**int binarySearch(float arr[], int n) {**

**int low = 0, high = n - 1, result = -1;**

**while (low <= high) {**

**int mid = (low + high) / 2;**

**if (arr[mid] >= 30.0) {**

**result = mid;**

**high = mid - 1;**

**} else {**

**low = mid + 1;**

**}**

**}**

**return result;**

**}**

**int main() {**

**srand(time(0)); // ✅ Seed for randomness**

**int n = 100;**

**float temp[n];**

**generateRandomSortedTemps(temp, n);**

**// Linear Search**

**clock\_t start = clock();**

**int linIndex = linearSearch(temp, n);**

**clock\_t end = clock();**

**double linTime = (double)(end - start) / CLOCKS\_PER\_SEC;**

**// Binary Search**

**start = clock();**

**int binIndex = binarySearch(temp, n);**

**end = clock();**

**double binTime = (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("Task C - Random Sorted Temperature >= 30°C\n\n");**

**printf("Linear Search: Index = %d, Value = %.2f°C, Time = %lf seconds\n",**

**linIndex, temp[linIndex], linTime);**

**printf("Binary Search: Index = %d, Value = %.2f°C, Time = %lf seconds\n",**

**binIndex, temp[binIndex], binTime);**

**return 0;**

**}**

**Output :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tasks** |  | **N=10^2** | **N=10^4** | **N=10^6** |
|  | **Linear Search** |  |  | **-** |
|  | **Binary Search** |  |  | **-** |

**Conclusion :**

**🔹 Task A – Linear Search**

* **Approach: Traverse array once to find min temperature and max pressure.**
* **Time Complexity: O(n)**
* **Observation: Fast even for large n like 1,000,000.**
* **Conclusion: Efficient and simple; suitable for large datasets.**

**🔹 Task B – Naive Pairwise Comparison**

* **Approach: Each element is compared with all others to find min/max.**
* **Time Complexity: O(n²)**
* **Observation: Very slow for large n; practical only for small values like n = 100.**
* **Conclusion: Inefficient due to quadratic comparisons; not recommended for big data.**

**🔹 Task C – Linear vs Binary Search**

* **Approach: Search for first temperature ≥ 30°C in sorted array.**
* **Time Complexity:**
  + **Linear: O(n)**
  + **Binary: O(log n)**
* **Observation: Binary search is significantly faster but both may appear too fast for time measurement when n is small.**
* **Conclusion: Use binary search for sorted data; repeat search multiple times for accurate timing with large n.**