**Task Algorithms**

**1)(Recursive & merge sort)**

algorithm mergeSort(arr, p, n) O(n log n)

// arr is an array of size n

{

if p < n then

q←floor((p+r)/2)

mergeSort(arr, p, q)

mergeSort(arr, q+1, n)

merge(arr, p, q, n)

}

algorithm checkTriangle(arr, n) O(n log n)

{

call mergeSort (arr, p, n) O(n log n)

for i ← 0 to n - 3 do O(n)

if arr[i] + arr[i + 1] > arr[i + 2] then

return 1

return 0

}

algorithm main() O(n log n)

{

input n

declare array arr of size n

for i ← 0 to n - 1 do O(n)

input arr[i]

result ← checkTriangle(arr, n) O(n log n)

output result

}

**Time Complexity = O(n log n)**

**2)(Non Recursive & bubble sort)**

algorithm isTriangle(arr, n) O(n^2)

{

// Bubble Sort O(n^2)

for i ← 0 to n - 2 do

for j ← 0 to n - i - 2 do

if arr[j] > arr[j + 1] then

temp ← arr[j]

arr[j] ← arr[j + 1]

arr[j + 1] ← temp

for i ← 0 to n - 3 do O(n)

a ← arr[i]

b ← arr[i + 1]

c ← arr[i + 2]

if (a + b > c) and (b + c > a) and (c + a > b) then

return 1

return 0

}

algorithm main() O(n^2)

{

input n

declare array arr of size n

for i ← 0 to n - 1 do O(n)

input arr[i]

result ← isTriangle(arr, n) (n^2)

output result

}

**Time Complexity = O(n^2)**

**3)(Recursive)**

algorithm checkTriangle(arr, n, p, q, r) O(n^3)

// arr is an array of size n

{

if p ≥ n - 2 then O(1)

return 0

if q ≥ n - 1 then O(n)

return checkTriangle(arr, n, p + 1, p + 2, p + 3)

if r ≥ n then O(n)

return checkTriangle(arr, n, p, q + 1, q + 2)

a ← arr[p]

b ← arr[q]

c ← arr[r]

if (a + b > c) and (b + c > a) and (c + a > b) then O(n^3)

return 1

return checkTriangle(arr, n, p, q, r + 1)

}

algorithm main() O(n^3)

{

input n

declare array arr of size n

for i ← 0 to n - 1 do O(n)

input arr[i]

result ← checkTriangle(arr, n, 0, 1, 2) O(n^3)

output result

}

**Time Complexity = O(n^3)**

**4)(Non Recursive)**

algorithm isTriangle(arr, n) O(n^3)

{

for p ← 0 to n - 3 do O(n)

for q ← p + 1 to n - 2 do O(n)

for r ← q + 1 to n - 1 do O(n)

a ← arr[p]

b ← arr[q]

c ← arr[r]

if (a + b > c) and (b + c > a) and (c + a > b) then O(1)

return 1

return 0

}

algorithm main() O(n^3)

{

input n

declare array arr of size n

for i ← 0 to n - 1 do O(n)

input arr[i]

result ← isTriangle(arr, n) O(n^3)

output result

}

**Time Complexity = O(n^3)**

**Comparison**

**1)(Recursive & merge sort):**

**- Description:** Sorts array using merge sort, then checks if any 3 consecutive elements form a triangle

-**Time Complexity:** O(n log n)

**-Best Case:** O(n log n)

**-Worst Case:** O(n log n)

**- Advantages:**

**-** Efficient

- Uses sorting to reduce checks

- Good for large arrays

**- Disadvantages:**

**-** Uses recursion (may lead to stack overflow for huge arrays)

**2)(Non Recursive & bubble sort):**

**-Description :** Sorts array using bubble sort, then checks triangle condition

-**Time Complexity:** O(n^2)

**-Best Case:** O(n) if sorted

**-Worst Case:**O(n^2)

**- Advantages:**

- Simple to understand

- No recursion

**- Disadvantages:**

- Inefficient for large arrays

- Sorting is slow

**3)(Recursive):**

**-Description :** Recursively checks all triplets in the array without sorting

-**Time Complexity:** O(n^3)

**-Best Case:** O(1) if triangle found early

**-Worst Case:** O(n^3)

**- Advantages:**

- Easy to implement

- Doesn't require sorting

**- Disadvantages:**

- Very slow for large arrays

- High memory if recursion depth is large

**4)(Non Recursive):**

**-Description :** Uses 3 nested loops to check all possible triplets

-**Time Complexity:** O(n^3)

**-Best Case:** O(1) if triangle found early

**-Worst Case:** O(n^3)

**- Advantages:**

- Very straightforward

- No recursion or sorting

**- Disadvantages:**

- Extremely slow

- Not efficient for large data

**Conclusion: Among all methods, the merge sort-based recursive approach is the most efficient for large datasets due to its O(n log n) time complexity.**

**Implementation by C:**

**1)(Recursive & merge sort):**

#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int\* L = (int\*) malloc(n1 \* sizeof(int));

int\* R = (int\*) malloc(n2 \* sizeof(int));

for (int i = 0; i < n1; i++) L[i] = arr[l + i];

for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

int i = 0, j = 0, k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) arr[k++] = L[i++];

else arr[k++] = R[j++];

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

free(L);

free(R);

}

// merge sort

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int checkTriangleRecursive(int arr[], int i, int n) {

if (i > n - 3) return 0;

if (arr[i] + arr[i + 1] > arr[i + 2]) return 1;

return checkTriangleRecursive(arr, i + 1, n);

}

int isTriangleRecursive(int arr[], int n) {

if (n < 3) return 0;

mergeSort(arr, 0, n - 1);

return checkTriangleRecursive(arr, 0, n);

}

int main() {

int n;

printf("Enter number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements:\n");

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

scanf("%d", &arr[i]);

int result = isTriangleRecursive(arr, n);

printf("Output: %d\n", result);

return 0;

}

**2)(Non Recursive & bubble sort):**

#include <stdio.h>

#include <stdlib.h>

int isTriangle(int arr[], int n) {

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

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

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

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

for (int i = 0; i < n - 2; i++) {

int a = arr[i];

int b = arr[i + 1];

int c = arr[i + 2];

if (a + b > c && b + c > a && c + a > b) {

return 1;

}

}

return 0;

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements:\n");

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

scanf("%d", &arr[i]);

}

int result = isTriangle(arr, n);

printf("Output: %d\n", result);

return 0;

}

**3)(Recursive):**

#include <stdio.h>

int checkTriangleRecursive(int arr[], int n, int p, int q, int r) {

if (p >= n - 2) return 0;

if (q >= n - 1) return checkTriangleRecursive(arr, n, p + 1, p + 2, p + 3);

if (r >= n) return checkTriangleRecursive(arr, n, p, q + 1, q + 2);

int a = arr[p], b = arr[q], c = arr[r];

if (a + b > c && b + c > a && c + a > b)

return 1;

return checkTriangleRecursive(arr, n, p, q, r + 1);

}

int main() {

int n;

printf("Enter number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements:\n");

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

scanf("%d", &arr[i]);

int result = checkTriangleRecursive(arr, n, 0, 1, 2);

printf("Output: %d\n", result);

return 0;

}

**4)(Non Recursive):**

#include <stdio.h>

int isTriangle(int arr[], int n) {

for (int p = 0; p < n - 2; p++) {

for (int q = p + 1; q < n - 1; q++) {

for (int r = q + 1; r < n; r++) {

int a = arr[p], b = arr[q], c = arr[r];

if (a + b > c && b + c > a && c + a > b) {

return 1;

}

}

}

}

return 0;

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements:\n");

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

scanf("%d", &arr[i]);

}

int result = isTriangle(arr, n);

printf("Output: %d\n", result);

return 0;

}