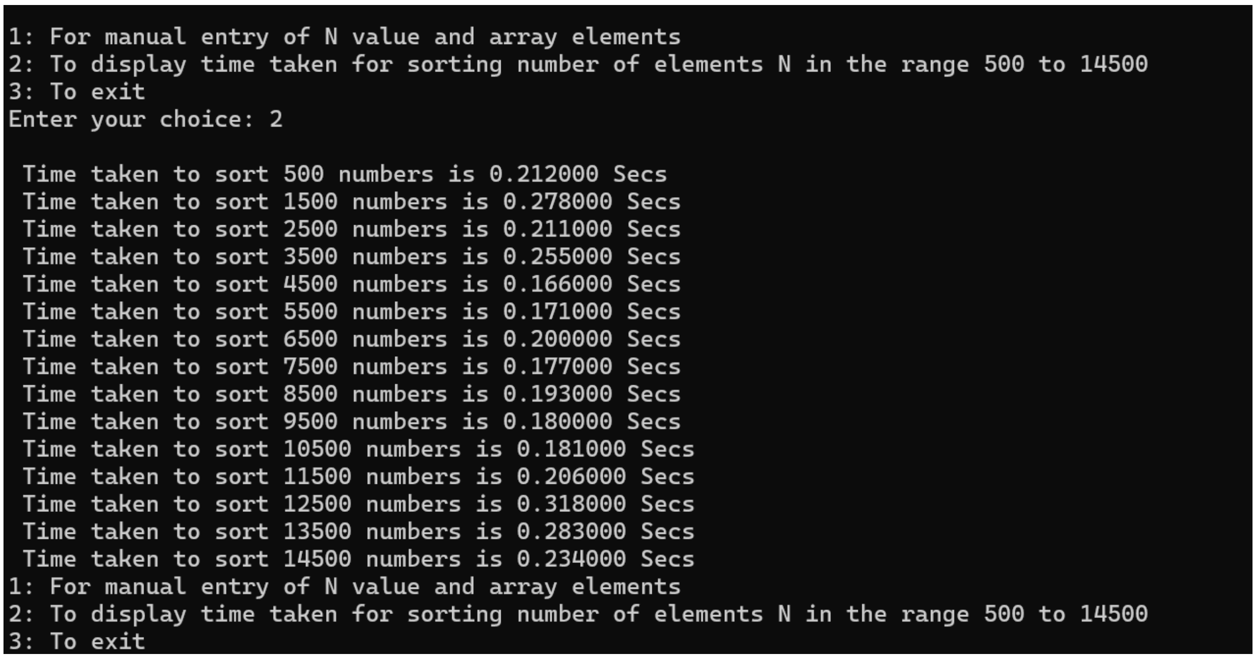
LAB-8

Question :

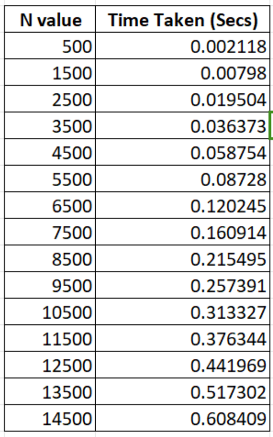
1. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.
2. Implement All Pair Shortest paths problem using Floyd's algorithm.

1.SOURCE CODE:

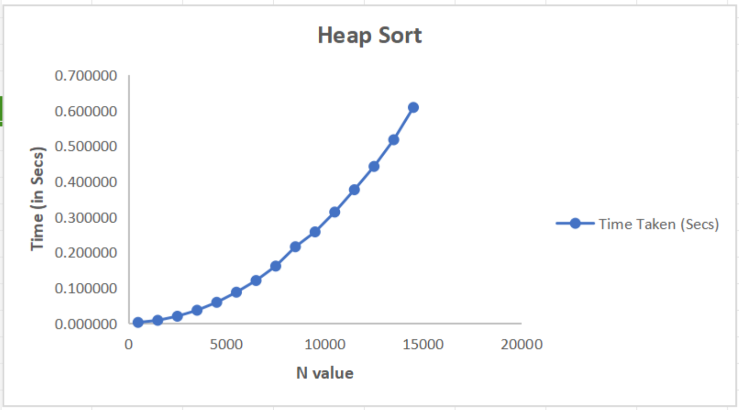
```
#include <stdio.h>
#include <time.h>

void swap(int* a, int* b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

// Function to heapify a subtree rooted at index i in arr[]
void heapify(int arr[], int n, int i) {
  int largest = i;  // Initialize largest as root
  int l = 2 * i + 1;  // left = 2*i + 1
  int r = 2 * i + 2;  // right = 2*i + 2

  // If left child is larger than root
  if (l < n && arr[l] > arr[largest])
    largest = l;

  // If right child is larger than largest so far
  if (r < n && arr[r] > arr[largest])
    largest = r;

  // If largest is not root
  if (largest != i) {
    swap(&arr[i], &arr[largest]);

    // Recursively heapify the affected sub-tree
    heapify(arr, n, largest);
    }
}

// Main function to do heap sort
void heapSort(int arr[], int n) {
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        swap(&arr[0], &arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

int main() {
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;

    while (1) {
        printf("\n1: For manual entry of N value and array elements");
        printf("\n2: To display time taken for sorting number of elements N in
the range 500 to 14500");
        printf("\n3: To exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("\nEnter the number of elements: ");
                scanf("%d", &n);
                printf("\nEnter array elements: ");
                for (i = 0; i < n; i++) {
                    scanf("%d", &a[i]);
                }
                start = clock();
                heapSort(a, n);
                end = clock();
                printf("\nSorted array is: ");
                for (i = 0; i < n; i++)
                    printf("%d\t", a[i]);
                printf("\n Time taken to sort %d numbers is %f Secs", n,
(((double)(end - start)) / CLOCKS_PER_SEC));
                break;
            case 2:
                n = 500;
                while (n <= 14500) {
                    for (i = 0; i < n; i++) {
                       a[i] = n - i;
                    }
                    start = clock();
                    heapSort(a, n);
                    for (j = 0; j < 500000000; j++) { temp = 38 / 600; }
                    end = clock();
                    printf("\n Time taken to sort %d numbers is %f Secs", n,
(((double)(end - start)) / CLOCKS_PER_SEC));
                    n = n + 1000;
                }
                break;
            case 3:
                exit(0);
        }
        getchar();
    }
}
```
RESULT:



```
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 2

 Time taken to sort 500 numbers is 0.212000 Secs
 Time taken to sort 1500 numbers is 0.278000 Secs
 Time taken to sort 2500 numbers is 0.211000 Secs
 Time taken to sort 3500 numbers is 0.255000 Secs
 Time taken to sort 4500 numbers is 0.166000 Secs
 Time taken to sort 5500 numbers is 0.171000 Secs
 Time taken to sort 6500 numbers is 0.200000 Secs
 Time taken to sort 7500 numbers is 0.177000 Secs
 Time taken to sort 8500 numbers is 0.193000 Secs
 Time taken to sort 9500 numbers is 0.180000 Secs
 Time taken to sort 10500 numbers is 0.181000 Secs
 Time taken to sort 11500 numbers is 0.206000 Secs
 Time taken to sort 12500 numbers is 0.318000 Secs
 Time taken to sort 13500 numbers is 0.283000 Secs
 Time taken to sort 14500 numbers is 0.234000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
```



N value	Time Taken (Secs)
500	0.002118
1500	0.00798
2500	0.019504
3500	0.036373
4500	0.058754
5500	0.08728
6500	0.120245
7500	0.160914
8500	0.215495
9500	0.257391
10500	0.313327
11500	0.376344
12500	0.441969
13500	0.517302
14500	0.608409

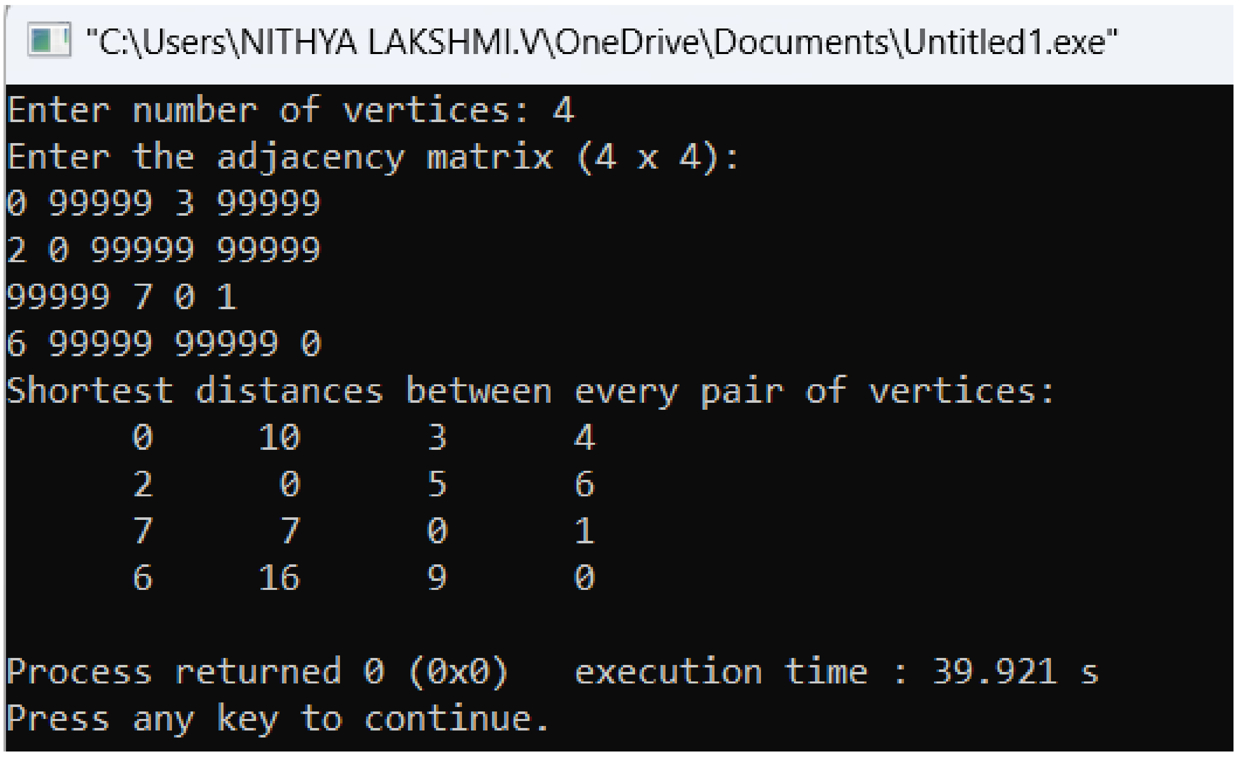


2.SOURCE CODE:

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

// Function to print the solution matrix
void printSolution(int V, int dist[][V]) {
  printf("Shortest distances between every pair of vertices:\n");
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
         if (dist[i][j] == INT_MAX)
            printf("%7s", "INF");
         else
            printf("%7d", dist[i][j]);
      }
      printf("\n");
   }
}

// Function to perform Floyd's algorithm to find shortest paths between all
pairs
void floydWarshall(int V, int graph[][V]) {
   int dist[V][V];  // Output matrix that will have the shortest distances
between every pair of vertices

   // Initialize dist[][] with the same values as graph[][]
   for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
         dist[i][j] = graph[i][j];

   // Iterate through all vertices as intermediate nodes
   for (int k = 0; k < V; k++) {
      // Pick all vertices as source one by one
      for (int i = 0; i < V; i++) {
         // Pick all vertices as destination for the above picked source
         for (int j = 0; j < V; j++) {
            // If vertex k is on the shortest path from i to j, then update dist[i][j]
            if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] +
dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
         }
      }
   }

   // Print the shortest distance matrix
   printSolution(V, dist);
}
```

```
int main() {
   int V;

   // Get number of vertices from user
   printf("Enter number of vertices: ");
   scanf("%d", &V);

   // Declare adjacency matrix graph[V][V]
   int graph[V][V];

   // Get the adjacency matrix from user
   printf("Enter the adjacency matrix (%d x %d):\n", V, V);
   for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
         scanf("%d", &graph[i][j]);
         if (graph[i][j] == 0 && i != j)  // Replace 0 with INF for non-diagonal 
elements
            graph[i][j] = INT_MAX;
      }
   }

   // Call the Floyd's algorithm function
   floydWarshall(V, graph);

   return 0;
}
```

RESULT:


"C:\Users\NITHYA LAKSHMI.V\OneDrive\Documents\Untitled1.exe"

```
Enter number of vertices: 4
Enter the adjacency matrix (4 x 4):
0 99999 3 99999
2 0 99999 99999
99999 7 0 1
6 99999 99999 0
Shortest distances between every pair of vertices:
      0     10       3      4
      2      0       5      6
      7      7       0      1
      6     16       9      0

Process returned 0 (0x0)   execution time : 39.921 s
Press any key to continue.
```