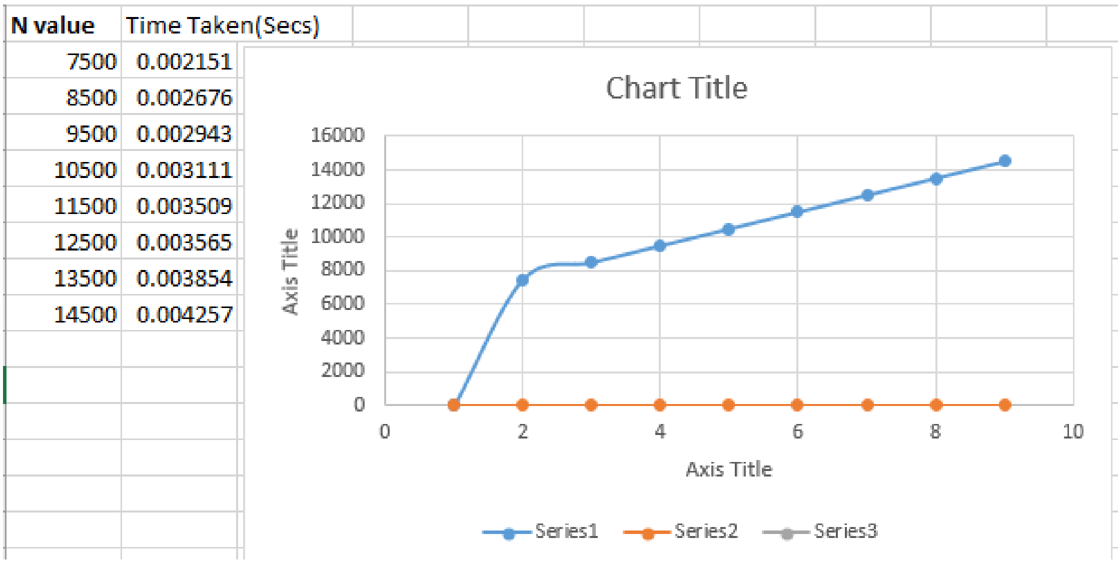
A given set of N integer elements using Heap Sort technique and compute its time taken.

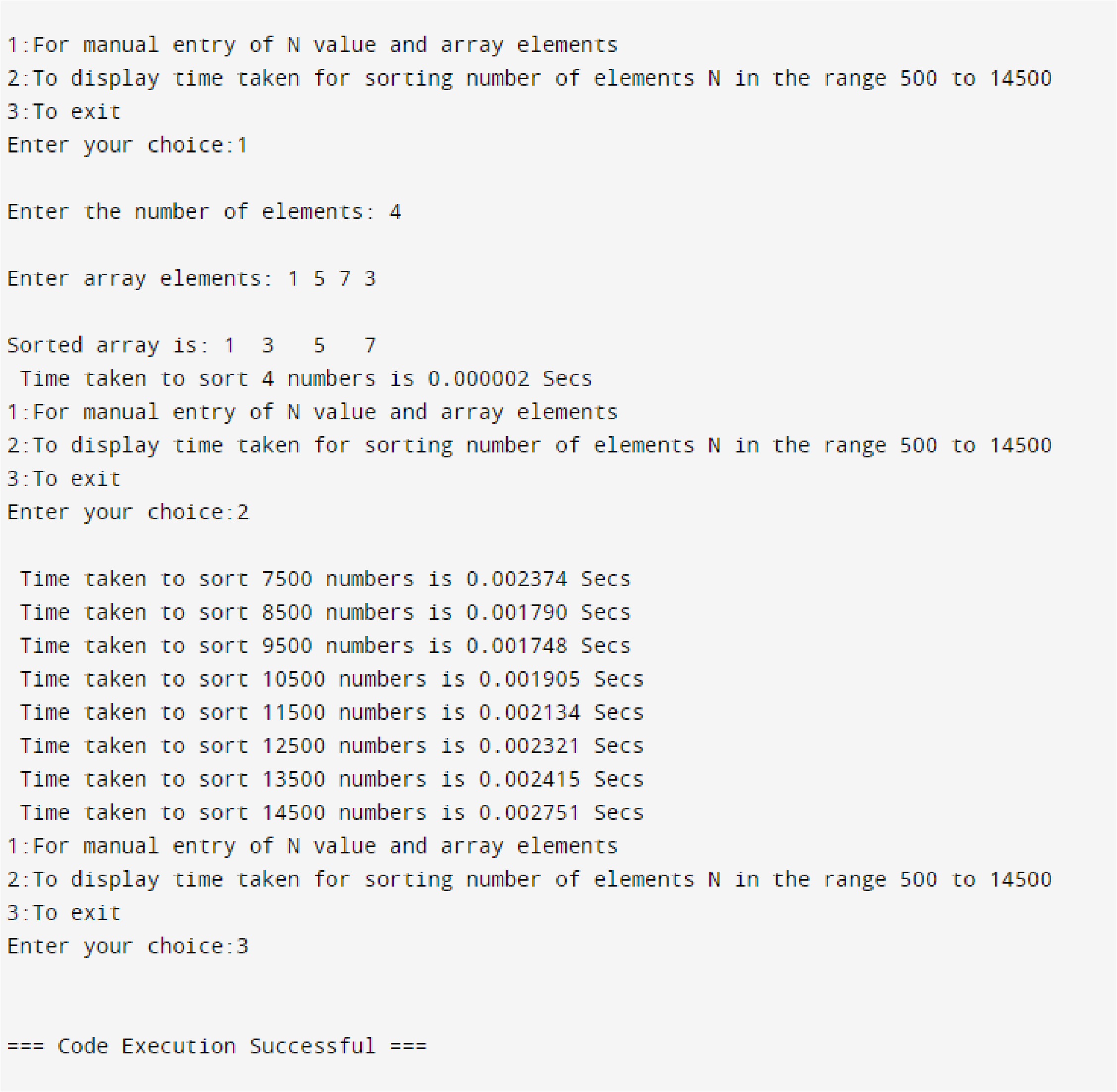
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
void swap(int* a, int* b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}
void heapify(int arr[], int N, int i)
{
   int largest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;
   if (left < N && arr[left] > arr[largest])
      largest = left;
   if (right < N && arr[right] > arr[largest])
      largest = right;
   if (largest != i) {
      swap(&arr[i], &arr[largest]);
      heapify(arr, N, largest);
   }
}
void heapSort(int arr[], int N)
{
   for (int i = N / 2 - 1; i >= 0; i--)
      heapify(arr, N, i);
   for (int i = N - 1; i >= 0; i--) {
      swap(&arr[0], &arr[i]);
```

```
        heapify(arr, i, 0);
    }
}
void main(){
   int a[100000],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=7500;
   while(n<=14500) {
      for(i=0;i<n;i++){
        a[i]=n-i;
       }
       start=clock();
       heapSort(a,n);
       for(j=0;j<500000;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();
  }
}
```

Output



N value	Time Taken(Secs)
7500	0.002151
8500	0.002676
9500	0.002943
10500	0.003111
11500	0.003509
12500	0.003565
13500	0.003854
14500	0.004257

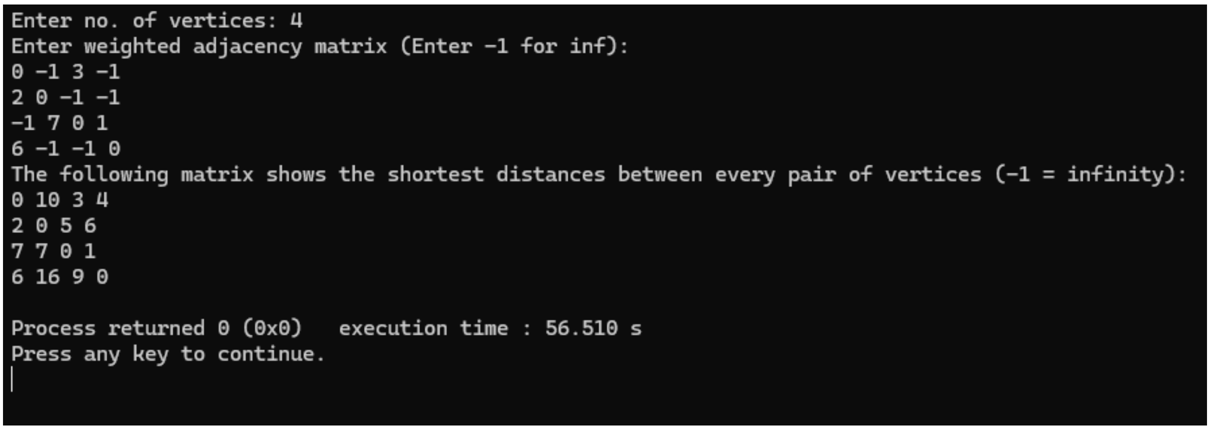


```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 4

Enter array elements: 1 5 7 3

Sorted array is: 1   3    5    7
 Time taken to sort 4 numbers is 0.000002 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
Enter your choice:2

 Time taken to sort 7500 numbers is 0.002374 Secs
 Time taken to sort 8500 numbers is 0.001790 Secs
 Time taken to sort 9500 numbers is 0.001748 Secs
 Time taken to sort 10500 numbers is 0.001905 Secs
 Time taken to sort 11500 numbers is 0.002134 Secs
 Time taken to sort 12500 numbers is 0.002321 Secs
 Time taken to sort 13500 numbers is 0.002415 Secs
 Time taken to sort 14500 numbers is 0.002751 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
Enter your choice:3


=== Code Execution Successful ===
```

Implement All Pair Shortest paths problem using Floyd’s algorithm

```
#include <stdio.h>

#include <limits.h>

int INF = 1e5;

void printSolution(int v, int dist[v][v]) {

    printf("The following matrix shows the shortest distances between every pair of vertices (-1 = infinity):\n");

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

        for (int j = 0; j < v; j++) {

            if (dist[i][j] == INF)

                printf("-1 ");

            else

                printf("%d ", dist[i][j]);

        }

        printf("\n");

    }

}

void floydWarshall(int v, int graph[v][v]) {

    int dist[v][v], i, j, k;

    for (i = 0; i < v; i++)

        for (j = 0; j < v; j++)

            dist[i][j] = graph[i][j];

    for (k = 0; k < v; k++) {

        for (i = 0; i < v; i++) {

            for (j = 0; j < v; j++) {

                if (dist[i][k] + dist[k][j] < dist[i][j])

                    dist[i][j] = dist[i][k] + dist[k][j];

            }

        }

    }
    printSolution(v, dist);
}
int main() {
    int v;
    printf("Enter no. of vertices: ");
    scanf("%d", &v);
    int graph[v][v];
    printf("Enter weighted adjacency matrix (Enter -1 for inf): \n");
    for(int i = 0; i < v; i++){
        for(int j = 0; j < v; j++){
            scanf("%d", &graph[i][j]);
            if (graph[i][j] == -1) graph[i][j] = INF;
        }
    }
    floydWarshall(v, graph);
    return 0;
}
```

Output



```
Enter no. of vertices: 4
Enter weighted adjacency matrix (Enter -1 for inf):
0 -1 3 -1
2 0 -1 -1
-1 7 0 1
6 -1 -1 0
The following matrix shows the shortest distances between every pair of vertices (-1 = infinity):
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0

Process returned 0 (0x0)   execution time : 56.510 s
Press any key to continue.
```