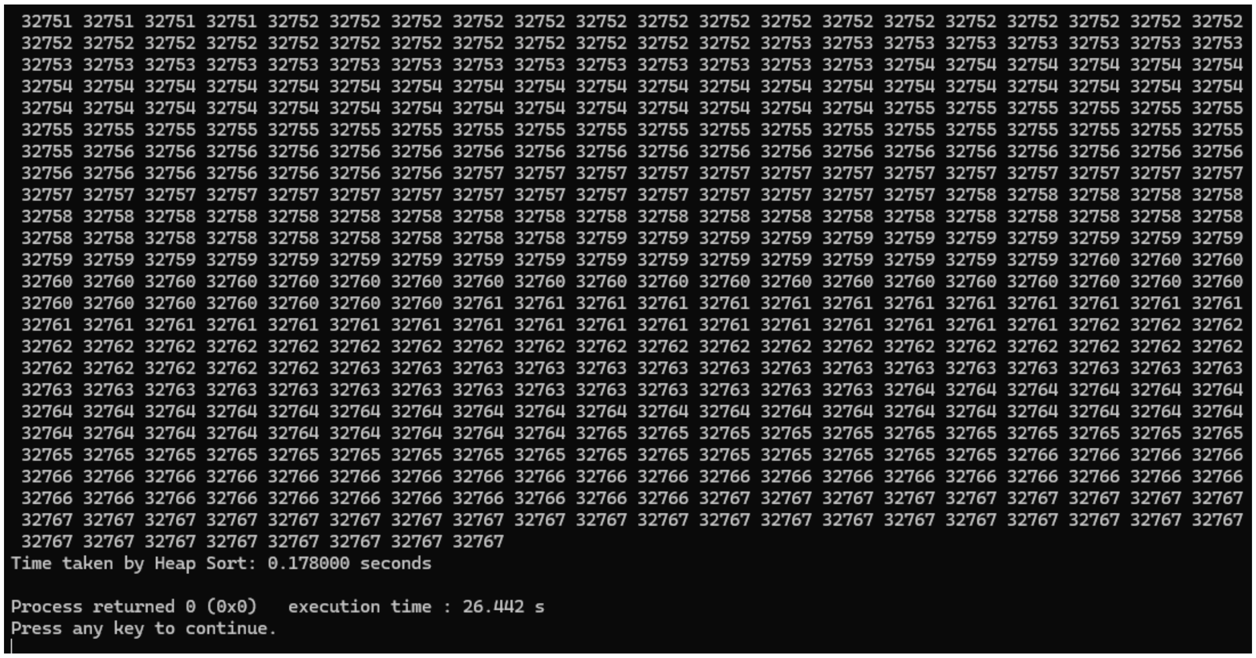
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

Code:

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

// Function to heapify a subtree rooted with node i
void heapify(int arr[], int n, int i) {
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left child
    int right = 2 * i + 2; // right child

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

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

    // If largest is not root
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

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

// Main function to perform 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);

    // Extract elements from heap one by one
    for (int i = n - 1; i >= 0; i--) {
        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

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

// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; ++i)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int n;
    n=1000000;
    //printf("Enter number of elements: ");
    //scanf("%d", &n);

    int *arr = (int *)malloc(n * sizeof(int));

    //printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; ++i) arr[i]=rand()%1000000;
        //scanf("%d", &arr[i]);

    clock_t start, end;
    double time_taken;

    start = clock();
    heapSort(arr, n);
    end = clock();

    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("Sorted array:\n");
    printArray(arr, n);

    printf("Time taken by Heap Sort: %f seconds\n", time_taken);

    free(arr);
    return 0;
}
```

Output:



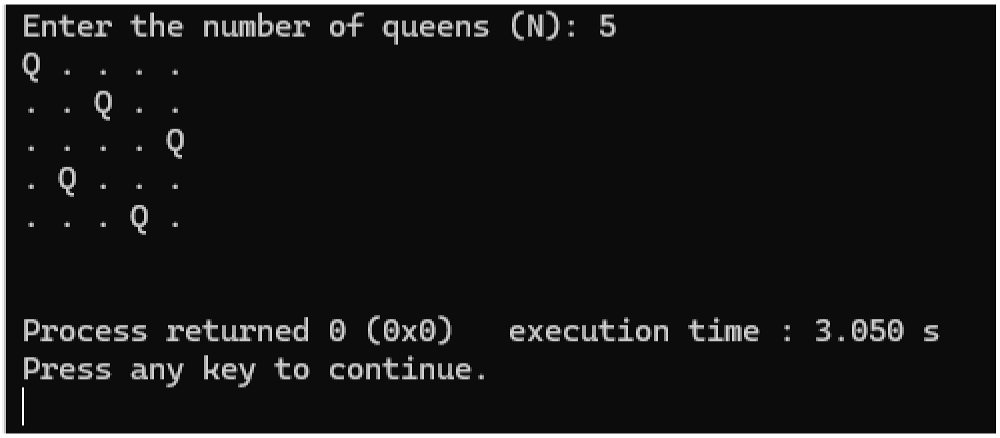
```
 32751 32751 32751 32751 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752
 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32752 32753 32753 32753 32753 32753 32753 32753 32753
 32753 32753 32753 32753 32753 32753 32753 32753 32753 32753 32753 32753 32753 32753 32754 32754 32754 32754 32754 32754
 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754
 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32754 32755 32755 32755 32755 32755 32755
 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755 32755
 32755 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756 32756
 32756 32756 32756 32756 32756 32756 32756 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757
 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32757 32758 32758 32758 32758 32758
 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758 32758
 32758 32758 32758 32758 32758 32758 32758 32758 32758 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759
 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32759 32760 32760 32760
 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760 32760
 32760 32760 32760 32760 32760 32760 32760 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761
 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32761 32762 32762 32762
 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762 32762
 32762 32762 32762 32762 32762 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763
 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32763 32764 32764 32764 32764 32764 32764
 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764 32764
 32764 32764 32764 32764 32764 32764 32764 32764 32764 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765
 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32765 32766 32766 32766 32766
 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766
 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32766 32767 32767 32767 32767 32767 32767 32767 32767
 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767 32767
 32767 32767 32767 32767 32767 32767 32767 32767
Time taken by Heap Sort: 0.178000 seconds

Process returned 0 (0x0)   execution time : 26.442 s
Press any key to continue.
```

Implement “N-Queens Problem” using Backtracking.

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

#define MAX 20

int board[MAX];
int N;

// Function to print the solution
void printSolution() {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i] == j)
                printf("Q ");
            else
                printf(". ");
        }
        printf("\n");
    }
    printf("\n");
}
```

```
// Function to check if a queen can be placed at board[row][col]
bool isSafe(int row, int col) {
    for (int i = 0; i < row; i++) {
        if (board[i] == col ||
            board[i] - i == col - row ||
            board[i] + i == col + row)
            return false;
    }
    return true;
}

// Recursive utility to solve N-Queens problem
bool solveNQueens(int row) {
    if (row == N) {
        printSolution();
        return true; // Return true to find only one solution
        // return false; // Uncomment this line and comment above to find all solutions
    }

    for (int col = 0; col < N; col++) {
        if (isSafe(row, col)) {
            board[row] = col;
            if (solveNQueens(row + 1))
                return true; // Return true to stop after first solution
        }
    }
    return false;
}

int main() {
    printf("Enter the number of queens (N): ");
    scanf("%d", &N);

    if (N <= 0 || N > MAX) {
        printf("Invalid value of N. Please use N between 1 and %d\n", MAX);
        return 1;
    }

    if (!solveNQueens(0)) {
        printf("No solution exists for %d queens.\n", N);
    }

    return 0;
}
```

Output:



```
Enter the number of queens (N): 5
Q . . . .
. . Q . .
. . . . Q
. Q . . .
. . . Q .


Process returned 0 (0x0)   execution time : 3.050 s
Press any key to continue.
```