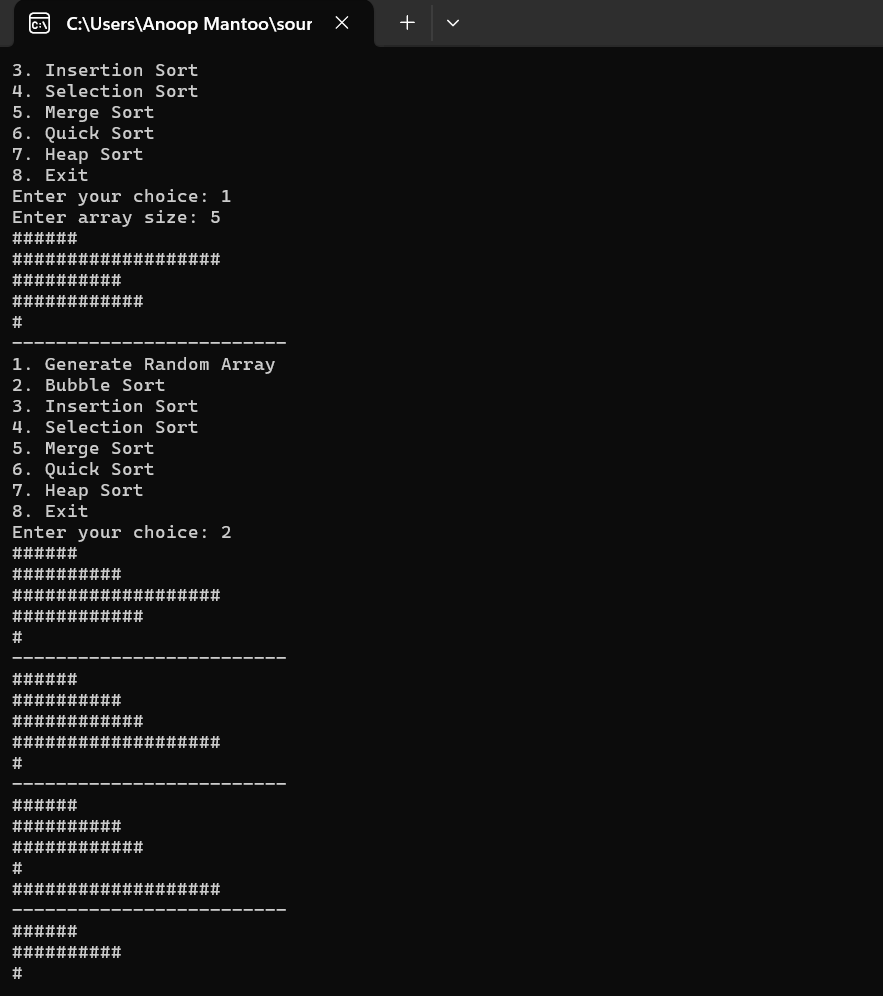
**OUTPUT:** ****

**SOURCE CODE FOR SORTING VISUALIZER:**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <thread>

#include <chrono>

using namespace std;

void printArray(const vector<int>& arr);

void bubbleSort(vector<int>& arr);

void insertionSort(vector<int>& arr);

void selectionSort(vector<int>& arr);

void mergeSort(vector<int>& arr, int left, int right);

void merge(vector<int>& arr, int left, int mid, int right);

void quickSort(vector<int>& arr, int low, int high);

int partition(vector<int>& arr, int low, int high);

void heapSort(vector<int>& arr);

void heapify(vector<int>& arr, int n, int i);

// Utility function to print the array

void printArray(const vector<int>& arr) {

for (int num : arr) {

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

cout << "#";

}

cout << endl;

}

cout << "-------------------------" << endl;

}

// Function to generate a random array

vector<int> generateRandomArray(int size) {

vector<int> arr(size);

for (int& num : arr) {

num = rand() % 20 + 1; // Values between 1 and 20

}

return arr;

}

// Function to pause for a given duration in seconds

void pause(int seconds) {

this\_thread::sleep\_for(chrono::seconds(seconds));

}

// Sorting algorithms implementations

void bubbleSort(vector<int>& arr) {

int n = arr.size();

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

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

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

swap(arr[j], arr[j + 1]);

printArray(arr);

pause(1); // Adjust speed as needed

}

}

}

}

void insertionSort(vector<int>& arr) {

int n = arr.size();

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

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

--j;

}

arr[j + 1] = key;

printArray(arr);

pause(1); // Adjust speed as needed

}

}

void selectionSort(vector<int>& arr) {

int n = arr.size();

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

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

swap(arr[i], arr[minIndex]);

printArray(arr);

pause(1); // Adjust speed as needed

}

}

void merge(vector<int>& arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> L(n1);

vector<int> R(n2);

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

for (int i = 0; i < n2; ++i) R[i] = arr[mid + 1 + i];

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

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++];

}

void mergeSort(vector<int>& arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

printArray(arr);

pause(1); // Adjust speed as needed

}

}

int partition(vector<int>& arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; ++j) {

if (arr[j] < pivot) {

swap(arr[++i], arr[j]);

}

}

swap(arr[i + 1], arr[high]);

return i + 1;

}

void quickSort(vector<int>& arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

printArray(arr);

pause(1); // Adjust speed as needed

}

}

void heapify(vector<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(vector<int>& arr) {

int n = arr.size();

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);

printArray(arr);

pause(1); // Adjust speed as needed

}

}

// Main function and controls

int main() {

srand(static\_cast<unsigned int>(time(0)));

int choice, size;

vector<int> arr;

while (true) {

cout << "1. Generate Random Array\n";

cout << "2. Bubble Sort\n";

cout << "3. Insertion Sort\n";

cout << "4. Selection Sort\n";

cout << "5. Merge Sort\n";

cout << "6. Quick Sort\n";

cout << "7. Heap Sort\n";

cout << "8. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 8) break;

switch (choice) {

case 1:

cout << "Enter array size: ";

cin >> size;

arr = generateRandomArray(size);

printArray(arr);

break;

case 2:

bubbleSort(arr);

break;

case 3:

insertionSort(arr);

break;

case 4:

selectionSort(arr);

break;

case 5:

mergeSort(arr, 0, arr.size() - 1);

break;

case 6:

quickSort(arr, 0, arr.size() - 1);

break;

case 7:

heapSort(arr);

break;

default:

cout << "Invalid choice!" << endl;

break;

}

}

return 0;

}