Name: Ankush Singh
Group: 5C13
Roll No.: 040

Date: 12-09-2023

EXPERIMENT- 1

Aim:

To implement following algorithm using array as a data structure and analyse its time complexity.

a) Merge sort

b) Quick sort

c) Bubble sort

d) Selection sort

e) Heap sort

Code:

```
#include <bits/stdc++.h>
#include <ctime>

using namespace std;

// Merge Sort
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int leftArr[n1];
    int rightArr[n2];

    for (int i = 0; i < n1; i++) {
        leftArr[i] = arr[left + i];
    }

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

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

    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }
```

Name: Ankush Singh
Group: 5C13
Roll No.: 040

Date: 12-09-2023

```
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}
void mergeSort(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);
    }
}
// Quick Sort
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i + 1], arr[high]);
    return (i + 1);
}
void quickSort(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);
    }
}
// Bubble Sort
void bubbleSort(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]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}
// Selection Sort
void selectionSort(int arr[], int n) {
    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]);
    }
}
```

Name: Ankush Singh
Group: 5C13
Roll No.: 040

Date: 12-09-2023

```
// Heap Sort
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);
    }
}
int main() {
    const int arrSize = 1000;
    int arr[arrSize], arrCopy[arrSize];

    // Fill the arrays with random values
    srand(time(NULL));
    for (int i = 0; i < arrSize; i++) {
        arr[i] = rand() % 1000;
    }

    int numTrials = 5;

    double totalMergeSortTime = 0.0;
    double totalQuickSortTime = 0.0;
    double totalBubbleSortTime = 0.0;
    double totalSelectionSortTime = 0.0;
    double totalHeapSortTime = 0.0;

    for (int trial = 0; trial < numTrials; ++trial) {
        copy(begin(arr), end(arr), begin(arrCopy));

        // Merge Sort
        clock_t start_time = clock();
        mergeSort(arrCopy, 0, arrSize - 1);
        clock_t end_time = clock();
        totalMergeSortTime += (end_time - start_time) * 1000.0 / CLOCKS_PER_SEC;

        // Quick Sort
        copy(begin(arr), end(arr), begin(arrCopy));
        start_time = clock();
        quickSort(arrCopy, 0, arrSize - 1);
        end_time = clock();
```

Name: Ankush Singh
Group: 5C13
Roll No.: 040

Date: 12-09-2023

```
        totalQuickSortTime += (end_time - start_time) * 1000.0 / CLOCKS_PER_SEC;

        // Bubble Sort
        copy(begin(arr), end(arr), begin(arrCopy));
        start_time = clock();
        bubbleSort(arrCopy, arrSize);
        end_time = clock();
        totalBubbleSortTime += (end_time - start_time) * 1000.0 / CLOCKS_PER_SEC;

        // Selection Sort
        copy(begin(arr), end(arr), begin(arrCopy));
        start_time = clock();
        selectionSort(arrCopy, arrSize);
        end_time = clock();
        totalSelectionSortTime += (end_time - start_time) * 1000.0 / CLOCKS_PER_SEC;

        // Heap Sort
        copy(begin(arr), end(arr), begin(arrCopy));
        start_time = clock();
        heapSort(arrCopy, arrSize);
        end_time = clock();
        totalHeapSortTime += (end_time - start_time) * 1000.0 / CLOCKS_PER_SEC;
    }

    double avgMergeSortTime = totalMergeSortTime / numTrials;
    cout << "Merge Sort Execution time: " << avgMergeSortTime << " milliseconds" << endl;

    double avgQuickSortTime = totalQuickSortTime / numTrials;
    cout << "Quick Sort Execution time: " << avgQuickSortTime << " milliseconds" << endl;

    double avgBubbleSortTime = totalBubbleSortTime / numTrials;
    cout << "Bubble Sort Execution time: " << avgBubbleSortTime << " milliseconds" <<
endl;

    double avgSelectionSortTime = totalSelectionSortTime / numTrials;
    cout << "Selection Sort Execution time: " << avgSelectionSortTime << " milliseconds"
<< endl;

    double avgHeapSortTime = totalHeapSortTime / numTrials;
    cout << "Heap Sort Execution time: " << avgHeapSortTime << " milliseconds" << endl;

    return 0;
}
```

Output:

```
Merge Sort Execution time: 0.1172 milliseconds
Quick Sort Execution time: 0.0976 milliseconds
Bubble Sort Execution time: 2.688 milliseconds
Selection Sort Execution time: 0.8404 milliseconds
Heap Sort Execution time: 0.1664 milliseconds
```