Assignment 3: Assignment based on analysis of quick sort (deterministic and randomized variant)

Name: Adnan Sadar

Roll No: 2

Batch:3

Problem Statement: Study the relative relation between execution time of best, avg, worst case scenarios of Quick sort

Code:

import java.util.\*;

class QuickSort {

public static *int* N = 5;

public static *int*[] arr = new *int*[N];

public static *void* main(*String* *args*[]) {

*Scanner* sc = new Scanner(System.in);

*Random* random = new Random();

System.out.println("Enter n: ");

*int* n = sc.nextInt();

*int* arr[] = new *int*[n];

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

arr[i] = random.nextInt(100000);

}

*long*[] time = new *long*[10];

*QuickSort* ob = new QuickSort();

// Randomised Quick Sort

System.out.println("Randomised Sort:");

for (*int* i = 0; i < 10; i++) {

*long* start = System.nanoTime();

ob.randomisedSort(arr, 0, n - 1);

// System.out.println("sorted array");

// printArray(arr);

*long* end = System.nanoTime();

time[i] = end - start;

}

*int* maxrandomised = 0, min = 0, avgrandomised = 0;

for (*int* i = 0; i < time.length; i++) {

System.out.println("Test Case " + (i + 1) + ": " + time[i] + "ns");

if (time[i] < time[min]) {

min = i;

}

if (time[i] > time[maxrandomised]) {

maxrandomised = i;

}

avgrandomised += time[i];

}

avgrandomised = avgrandomised / 10;

System.out.println("\nRandomised Sort:");

System.out.println("\nBest Case: " + time[min] + "ns");

System.out.println("Worst Case: " + time[maxrandomised] + "ns");

System.out.println("Average Case: " + avgrandomised + "ns\n");

// Deterministic Quick Sort

for (*int* i = 0; i < 10; i++) {

arr[i] = random.nextInt(100000);

*long* start = System.nanoTime();

ob.deterministicSort(arr, 0, n - 1);

// System.out.println("sorted array");

// printArray(arr);

*long* end = System.nanoTime();

time[i] = end - start;

}

*int* maxdeterministic = 0;

min = 0;

*int* avgdeterministic = (*int*) time[0];

for (*int* i = 0; i < time.length; i++) {

System.out.println("Test Case " + (i + 1) + ": " + time[i] + "ns");

if (time[i] < time[min]) {

min = i;

}

if (time[i] > time[maxdeterministic]) {

maxdeterministic = i;

}

avgdeterministic += time[i];

}

avgdeterministic = avgdeterministic / 10;

System.out.println("\nDeterministic Sort:");

System.out.println("\nBest Case: " + time[min] + "ns");

System.out.println("Worst Case: " + time[maxdeterministic] + "ns");

System.out.println("Average Case: " + avgdeterministic + "ns");

if (avgdeterministic < avgrandomised) {

System.out.println("Conclusion: Deterministic Quick Sort is faster than Randomised Quick Sort");

} else {

System.out.println("Conclusion: Randomised Quick Sort is faster than Deterministic Quick Sort");

}

sc.close();

}

*void* random(*int* *low*, *int* *high*) {

*Random* rand = new Random();

*int* pivotrand = rand.nextInt(high - low) + low;

*int* temp1 = arr[pivotrand];

arr[pivotrand] = arr[high];

arr[high] = temp1;

}

*int* randomisedPartition(*int* *arr*[], *int* *low*, *int* *high*) {

*int* pivotrand = arr[high];

*int* i = (low - 1); // index of smaller element

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

// If current element is smaller than or

// equal to pivot

if (arr[j] <= pivotrand) {

i++;

// swap arr[i] and arr[j]

*int* temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

// swap arr[i+1] and arr[high] (or pivot)

}

*int* temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

*void* randomisedSort(*int* *arr*[], *int* *low*, *int* *high*) {

if (low < high) {

*int* pi = randomisedPartition(arr, low, high);

randomisedSort(arr, low, pi - 1);

randomisedSort(arr, pi + 1, high);

}

}

*int* deterministicPartition(*int* *arr*[], *int* *low*, *int* *high*) {

*int* pivotdet = arr[high];

*int* i = (low - 1); // index of smaller element

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

// If current element is smaller than the pivot

if (arr[j] < pivotdet) {

i++;

// swap arr[i] and arr[j]

*int* temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// swap arr[i+1] and arr[high] (or pivot)

*int* temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

*void* deterministicSort(*int* *arr*[], *int* *low*, *int* *high*) {

if (low < high) {

*int* pi = deterministicPartition(arr, low, high);

deterministicSort(arr, low, pi - 1);

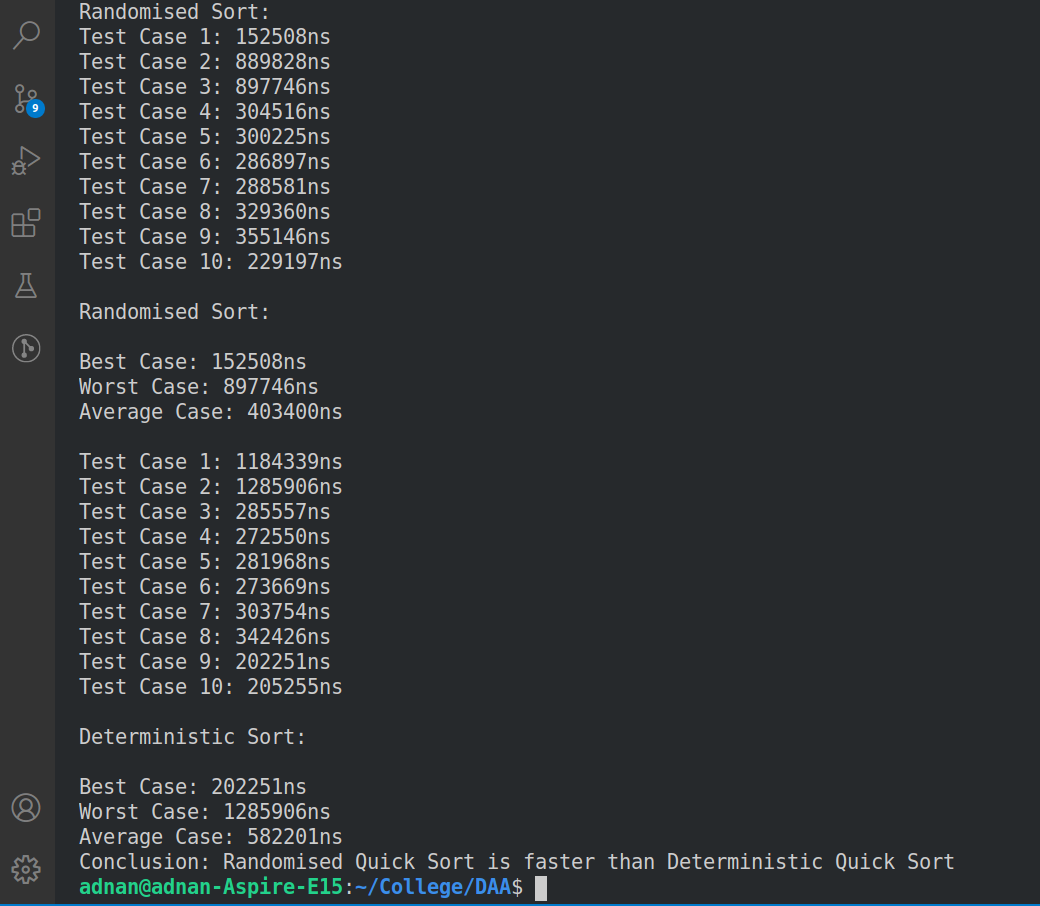
deterministicSort(arr, pi + 1, high);

}

}

}

Output:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N=5 | Deterministic Version | | Randomized Version | |
|  | **Best Case** | 0.003637083 seconds | **Best Case** | 0.001224225 seconds |
|  | **Worst Case** | 0.013391653 seconds | **Worst Case** | 0.018288269seconds |
|  | **Average Case** |  | **Average Case** |  |
| 1 |  | 0.003637083 seconds |  | 0.017575567 seconds |
| 2 |  | 0.012355236 seconds |  | 0.011473106 seconds |
| 3 |  | 0.011108090 seconds |  | 0.018288269 seconds |
| 4 |  | 0.011551223 seconds |  | 0.003972996 seconds |
| 5 |  | 0.013391653 seconds |  | 0.001301167 seconds |
| 6 |  | 0.010211706 seconds |  | 0.001255872 seconds |
| 7 |  | 0.010457824 seconds |  | 0.001230804 seconds |
| 8 |  | 0.010195822 seconds |  | 0.001224225 seconds |
| 9 |  | 0.009767666 seconds |  | 0.001264893 seconds |
| 10 |  | 0.006804531 seconds |  | 0.001297673 seconds |

Conclusion: Randomised Quick Sort performs better as compared to deterministic Quick Sort which is seen when the time complexities of both the approaches are compared side by side.

Problem Statement 2: Statistically verify that avg case time complexity of deterministic quicl sort is equivalent to expected time complexit of randomized quick sort based on the number of comparisons performed

Code:

//Randomised Quick Sort

import java.util.\*;

class QuickSortRComparisons {

static int N;

static int arr[];

int[][] T = new int[N][N];

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter N: ");

N = sc.nextInt();

int arr[] = new int[N];

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

arr[i] = i;

}

QuickSortRComparisons ob = new QuickSortRComparisons();

ob.sort(arr, 0, N - 1);

ob.comparisons();

// System.out.println("sorted array");

// printArray(arr);

sc.close();

}

void random(int low, int high) {

Random rand = new Random();

int pivot = rand.nextInt(high - low) + low;

int temp1 = arr[pivot];

arr[pivot] = arr[high];

arr[high] = temp1;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1); // index of smaller element

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

// If current element is smaller than or

// equal to pivot

if (arr[j] <= pivot) {

T[pivot][j] += 1;

T[j][pivot] += 1;

i++;

// swap arr[i] and arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// swap arr[i+1] and arr[high] (or pivot)

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

/\*

\* The main function that implements QuickSort() arr[] --> Array to be sorted,

\* low --> Starting index, high --> Ending index

\*/

void sort(int arr[], int low, int high) {

if (low < high) {

/\*

\* pi is partitioning index, arr[pi] is now at right place

\*/

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

// Recursively sort elements before

// partition and after partition

sort(arr, low, pi - 1);

sort(arr, pi + 1, high);

}

}

void comparisons() {

System.out.println("No of comparisons of pair i,j:");

System.out.println("00\t 01\t 02\t 03\t 04\t 05\t 06\t 07\t 08\t 09");

for (int i = 0; i < T.length; i++) {

for (int j = 0; j < T.length; j++) {

System.out.print(T[i][j]+"\t ");

}

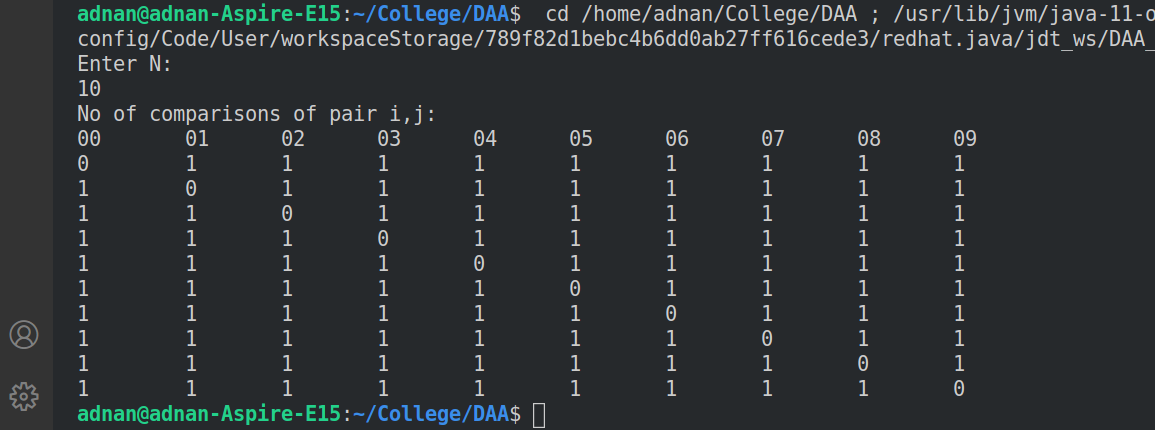
System.out.println();

}

}

}

Output:



//Deterministic Quick Sort

import java.util.\*;

class QuickSortDComparisons {

static int N;

static int arr[];

int[][] T = new int[N][N];

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

Random random = new Random();

System.out.println("Enter N: ");

N = sc.nextInt();

int arr[] = new int[N];

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

arr[i] = random.nextInt(100000);

}

QuickSortDComparisons ob = new QuickSortDComparisons();

ob.sort(arr, 0, N - 1);

ob.comparisons();

// System.out.println("sorted array");

// printArray(arr);

sc.close();

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1); // index of smaller element

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

// If current element is smaller than the pivot

if (arr[j] <= pivot) {

T[pivot][j] += 1;

T[j][pivot] += 1;

i++;

// swap arr[i] and arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// swap arr[i+1] and arr[high] (or pivot)

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

void sort(int arr[], int low, int high) {

if (low < high) {

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

sort(arr, low, pi - 1);

sort(arr, pi + 1, high);

}

}

void comparisons() {

System.out.println("No of comparisons of pair i,j:");

System.out.println("00\t 01\t 02\t 03\t 04\t 05\t 06\t 07\t 08\t 09");

for (int i = 0; i < T.length; i++) {

for (int j = 0; j < T.length; j++) {

System.out.print(T[i][j] + "\t ");

}

System.out.println();

}

}

}