**Quicksort & Randomized Quicksort Running Time Analysis**

**CS 560: Design & Analysis of Algorithms**

**Instructor: Kaushik Sinha**

**Graph 1:**

For the first graph, randomized quicksort T’’’(N) is the fastest, especially when the size of the array is larger (i.e. N = 10000). The regular quicksort, from the random list, follows the randomized quicksort from the random list in terms of time. Lastly, the slowest out of them, the randomized quicksort from the random list. From all of these, we can see the running times is greater than O(N).

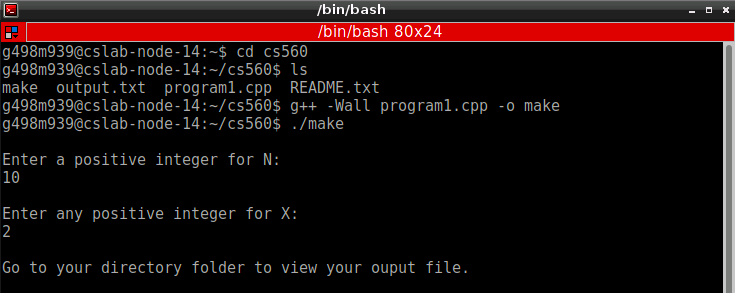
**Graph 2:**

T’’(N), quicksort in the normal sorted list, never reaches anywhere close to the running time of O(N^2). T’’(N) also takes longer compared to the rest, as it is the worst case for any quicksort method and it uses the highest index as its pivot.

**Code Description:**

With N and X given by user input, the function N+(i+1)\*X, is used to fill out the normal insertion arrays. In order to fill out the randomized array, we will use the rand function. The only difference between the randomized quicksort and the normal quicksort code is the partitioning. The normal partitioning chooses the highest index for the pivot, while the randomized partitioning will choose the pivot at random.

**Demonstration on CS Linux Server:**

****

**Appendix**

/\*

\* Name: Mubasshir Karim

\* WSU ID: G498M939

\* File: program1.cpp

\* Assignment: Programming Assignment

\* Description: Use Quicksort to sort N randomly generated integers

\* and then analyze various characteristics.

\*/

#include <chrono>

#include <ctime>

#include <fstream>

#include <iomanip>

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

// Setting the MAX\_RANGE to 10000

#define MAX\_RANGE 10000

// Normal Insertion

void normInsert(int arr[],int N, int X)

{

int i;

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

{

arr[i] = N+((i+1)\*X); // gets your normal array

} // N+1X, N+2X...

}

// Random Insertion

void randInsert(int arr[], int N)

{

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

{

arr[i] = rand()%(MAX\_RANGE); // gets your random array

} // rand function based on MAX\_RANGE

}

// Swap

void swap(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

// Normal Partition

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

{

int pivot = arr[high]; // Pivot

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

for (int j = low; j <= high- 1; j++) // If current element is smaller

{ // than or equal to pivot

if (arr[j] <= pivot)

{

i++; // Increment index of smaller element

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

}

}

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

return (i + 1);

}

// Random Partition

int randPartition(int arr[], int p, int r)

{

int pivotIndex = p + rand()%(r - p + 1);

int pivot;

int i = p - 1;

int j;

pivot = arr[pivotIndex];

swap(&arr[pivotIndex], &arr[r]);

for (j = p; j < r; j++)

{

if (arr[j] < pivot)

{

i++;

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

}

}

swap(&arr[i+1], &arr[r]);

return i + 1;

}

// Normal Quicksort

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

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

quickSort(arr, low, p - 1);

quickSort(arr, p + 1, high);

}

}

// Random Quicksort

void rqSort(int arr[], int low, int high)

{

int j;

if (low < high)

{

j = randPartition(arr, low, high);

rqSort(arr, low, j-1);

rqSort(arr, j+1, high);

}

}

// Main

int main()

{

int N, X;

ofstream outputFile ("output.txt");

cout << "\nEnter a positive integer for N:\n";

cin >> N;

cout << "\nEnter any positive integer for X:\n";

cin >> X;

int randArray[N], rsortArray[N],nsortArray[N];

int normArray[N], randsortNArray[N], quicksortNArray[N];

cout << "\nGo to your directory folder to view your ouput file.\n\n";

randInsert(randArray, N); // Random Insertion - to randomize the array

for (int i = 0; i < N; i++) // For loop to increment to user input

{

rsortArray[i] = randArray[i];

nsortArray[i] = randArray[i];

}

// Use clock to get run time

clock\_t ranRStart = clock();

rqSort(rsortArray, 0, N-1);

double rrTime = (clock() - ranRStart) / (double) CLOCKS\_PER\_SEC;

clock\_t quickRStart = clock();

quickSort(nsortArray, 0, N-1); // Quicksort of random array

double qrTime = (clock() - quickRStart) / (double) CLOCKS\_PER\_SEC;

normInsert(normArray,N,X); // Normal Insertion

for(int i=0;i< N; i++) // sorts Normal Insertion array

{

randsortNArray[i]=normArray[i];

quicksortNArray[i]=normArray[i];

}

//Time randomized Quicksort

clock\_t ranNStart = clock();

rqSort(randsortNArray, 0, N-1); // Noraml Array insertion of randomized sort

double rnTime = (clock() - ranNStart) / (double) CLOCKS\_PER\_SEC;

clock\_t quickNStart = clock();

quickSort(quicksortNArray, 0, N-1); // Normal Array insertion of quicksort

double qnTime = (clock() - quickNStart) / (double) CLOCKS\_PER\_SEC;

// Output Text File

if (outputFile .is\_open())

{

// Displays first table (Random Insertion)

outputFile << "Table 1: Random Insertion\n\n";

outputFile << "Random Array\t\t" << "Randomized Sort\t\t" << "Quick Sort\t\t\n";

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

{

outputFile << randArray[i] << setw(28);

outputFile << rsortArray[i] << setw(23);

outputFile << nsortArray[i] << endl;

}

outputFile << "\nRunning Time: Random Insertion\t" << "Randomized Sort\t"

<< rrTime << "\tQuick Sort\t" << qrTime << endl;

// Display second table (Normal Insertion)

outputFile << "\nTable 2: Normal Insertion (N+((i+1)\*x)\n\n";

outputFile << "Normal Array\t\t" << "Randomized Sort\t\t" << "Quick Sort\t\t\n";

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

{

outputFile << normArray[i] << setw(28);

outputFile << randsortNArray[i] << setw(23);

outputFile << quicksortNArray[i] << endl;

}

outputFile << "\nRunning Time: Normal Insertion\t" << "Randomized Sort\t" << rnTime

<< "\tQuick Sort\t" << qnTime << endl;

outputFile .close();

}

else cout << "Error occured in your output.txt file";

return 0;

}