**Fatma Kausar**

**CSC382**

**Lab 3 – QuickSort and Randomized QuickSort**

In this lab, you will implement Quick Sort and Randomized Quicksort algorithm for the same inputs.

For each algorithm, and for each n = 100, 200, 300, 400, 500, 1000, 4000, 10000, measure its running time and number of steps when the input is (1) already sort, i.e. n, n-1, …, 3, 2,1; (2) reversely sorted 1, 2, 3, … n; (3) random permutation of 1, 2, …, n; (4) 50 instances of n random numbers generated in the range of [1..n].

Note:

(1) You may have to repeat the algorithm many times, each time you need to initialize the array.

(2) Your running time should exclude the time for initialization.

(3) All measurement should be done in a single run, i.e. you do not need to run once for n=100, another time for n=200, etc

What to turn in:

(1) Well documented source code in C++

(2) Report the number of steps and the CPU running time in a table (Add this result in your previous table).

(3) Plot the running time of the algorithm results (Add this result in your previous plot).

Due on 11/17/2020

#include <iostream>

#include <string>

#include <time.h>

#include <vector>

#include <algorithm> //for random\_shuffle

#include <chrono> //for chronotime

#include <math.h> // for pow

using namespace std;

long long int stepCount = 0;

long double cApprox = 0;

//function for generating pivot

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

{

if (isRandom) {

int random = low + rand() % (high - low);

swap(arr[random], arr[high]);

}

int pivot = arr[high];

int i = (low - 1);

stepCount += 2;

for (int j = low; j <= high - 1; j++)

{

stepCount += 3;

if (arr[j] < pivot)

{

i++;

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

stepCount += 3;

}

}

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

stepCount += 1;

return (i + 1);

}

//The main function that implements QuickSort(Array to be sorted, Starting index, Ending index)

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

{

while (low < high) {

stepCount++;

int j = partition(arr, low, high, isRandom);

stepCount++;

if (j - low < high - j) {

stepCount += 1;

quickSort(arr, low, j - 1, isRandom);

stepCount++;

low = j + 1;

stepCount++;

}

else {

quickSort(arr, j + 1, high, isRandom);

stepCount++;

high = j - 1;

stepCount++;

}

}

}

//Printing the vector arrays function

void printArray(vector<int> &arr)

{

for (int x : arr)

{

cout << x << " ";

}

}

//Generating sorted arrays for n input

void generateSort(vector<int> &arr, int n)

{

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

{

arr.push\_back(i);

}

}

//Generate reverse arrays for n input

void generateReverse(vector<int> &arr, int n)

{

for (int i = n; i >= 1; i--)

{

arr.push\_back(i);

}

}

//Generate random permutation arrays for n input

void generateRandom(vector<int> &arr, int n)

{

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

{

arr.push\_back(i);

}

random\_shuffle(arr.begin(), arr.end());

}

int main()

{

// Array of inputs n

int inputs[8] = { 100, 200, 300, 400, 500, 1000, 4000, 10000 };

//QuickSort for Sorted Arrays, Reverse Arrays, Random Permutation Arrays

//C approximation will be steps/n^2

cout << "---------------------------------QUICK SORT----------------------------------------" << endl;

cout << "SORTED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateSort(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "REVERSED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateReverse(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "RANDOM PERMUTATION ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateRandom(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

//50 instances of random inputs quickSort

cout << "50 INSTANCES QUICK SORT" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

double totalT = 0;

for (int x = 0; x < 50; x++)

{

vector<int> arr;

for (int j = 1; j <= inputs[i]; j++)

{

arr.push\_back(rand() % inputs[i] + 1);

}

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

totalT += timer.count();

arr.clear();

}

cApprox = (stepCount / 50) / (pow(inputs[i], 2));

cout << "50 INSTANCES OF " << inputs[i] << " ELEMENTS: Steps Average = " << stepCount / 50 << " | Runtime = " << totalT / 50 << " miliseconds | Average time: " << totalT / 50 << " miliseconds. | C Approximation: " << cApprox << endl;

}

cout << "---------------------------------RANDOMIZED QUICK SORT----------------------------------------" << endl;

cout << "SORTED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateSort(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "REVERSED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateReverse(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "RANDOM PERMUTATION ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateRandom(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

//50 instances of random inputs quickSort

cout << "50 INSTANCES RANDOMIZED QUICK SORT" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

double totalT = 0;

for (int x = 0; x < 50; x++)

{

vector<int> arr;

for (int j = 1; j <= inputs[i]; j++)

{

arr.push\_back(rand() % inputs[i] + 1);

}

auto t1 = chrono::high\_resolution\_clock::now();

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

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

totalT += timer.count();

arr.clear();

}

cApprox = (stepCount / 50) / (pow(inputs[i], 2));

cout << "50 INSTANCES OF " << inputs[i] << " ELEMENTS: Steps Average = " << stepCount / 50 << " | Runtime = " << totalT / 50 << " miliseconds | Average time: " << totalT / 50 << " miliseconds. | C Approximation: " << cApprox << endl;

}

system("PAUSE");

return 0;

}

A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

**Quick Sort Running Time In Milliseconds**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 9.1574 | 3.6262 | 0.8123 | 0.446546 |
| 200 | 32.1684 | 14.451 | 1.8638 | 1.2174 |
| 300 | 69.0137 | 30.1637 | 2.8631 | 2.08805 |
| 400 | 90.6777 | 59.2476 | 2.8106 | 2.49517 |
| 500 | 132.495 | 95.6337 | 3.6629 | 3.59566 |
| 1000 | 551.195 | 350.687 | 9.8365 | 7.98307 |
| 4000 | 9772.05 | 5400.88 | 42.3004 | 38.9438 |
| 10000 | 58956.6 | 37429.1 | 110.206 | 114.318 |

**Quick Sort Steps**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 30393 | 22943 | 3407 | 3372 |
| 200 | 120793 | 90893 | 7711 | 7904 |
| 300 | 271193 | 203843 | 13116 | 12907 |
| 400 | 481593 | 361793 | 17547 | 18292 |
| 500 | 751993 | 564743 | 26014 | 23404 |
| 1000 | 3003993 | 2254493 | 55236 | 53108 |
| 4000 | 48015993 | 36017993 | 281328 | 267312 |
| 10000 | 300039993 | 225044993 | 752327 | 744941 |

**C Approximation Quick Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances  Average |
| 100 | 3.0393 | 2.2943 | 0.3407 | 0.3372 |
| 200 | 3.01982 | 2.27232 | 0.192775 | 0.1976 |
| 300 | 3.01326 | 2.26492 | 0.145733 | 0.143411 |
| 400 | 3.00996 | 2.26121 | 0.109669 | 0.114325 |
| 500 | 3.00797 | 2.25897 | 0.104056 | 0.093616 |
| 1000 | 3.00399 | 2.25449 | 0.055236 | 0.053108 |
| 4000 | 3.001 | 2.25112 | 0.017583 | 0.016707 |
| 10000 | 3.0004 | 2.25045 | 0.00752327 | 0.00744941 |

**Randomized Quick Sort Running Time In Milliseconds**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 0.4618 | 0.5611 | 0.883 | 0.56843 |
| 200 | 1.7321 | 1.6269 | 1.5783 | 1.23106 |
| 300 | 1.6617 | 2.2688 | 2.3511 | 2.38845 |
| 400 | 3.5778 | 3.1743 | 3.6188 | 3.00535 |
| 500 | 3.272 | 4.8072 | 4.5429 | 4.10966 |
| 1000 | 8.343 | 9.0569 | 9.5789 | 8.39373 |
| 4000 | 39.0535 | 43.2876 | 41.7797 | 41.9117 |
| 10000 | 114.882 | 100.187 | 110.731 | 119.848 |

**Randomized Quick Sort Steps**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 3263 | 3144 | 3853 | 3362 |
| 200 | 7944 | 7493 | 7132 | 7837 |
| 300 | 11683 | 13045 | 13970 | 13007 |
| 400 | 18053 | 18448 | 18007 | 18391 |
| 500 | 22082 | 22632 | 24288 | 23950 |
| 1000 | 50840 | 50777 | 55595 | 53632 |
| 4000 | 252102 | 265253 | 262103 | 268868 |
| 10000 | 728331 | 724450 | 754768 | 742219 |

**C Approximation Randomized Quick Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances  Average |
| 100 | 0.3263 | 0.3144 | 0.3853 | 0.3362 |
| 200 | 0.1986 | 0.187325 | 0.1783 | 0.195925 |
| 300 | 0.129811 | 0.144944 | 0.155222 | 0.144522 |
| 400 | 0.112831 | 0.1153 | 0.112544 | 0.114944 |
| 500 | 0.088328 | 0.090528 | 0.097152 | 0.0958 |
| 1000 | 0.05084 | 0.050777 | 0.055595 | 0.053632 |
| 4000 | 0.0157564 | 0.0165783 | 0.0163814 | 0.0168042 |
| 10000 | 0.00728331 | 0.0072445 | 0.00754768 | 0.00742219 |

**Heap Sort Running Time In Milliseconds**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 1.3209 | 1.1095 | 1.2338 | 1.25239 |
| 200 | 3.4804 | 2.6216 | 2.9825 | 2.9076 |
| 300 | 6.2832 | 4.3183 | 4.7049 | 5.53798 |
| 400 | 10.3561 | 6.2785 | 9.5601 | 8.86633 |
| 500 | 10.9865 | 11.8295 | 10.4293 | 9.79526 |
| 1000 | 24.9368 | 19.4624 | 23.0986 | 28.2599 |
| 4000 | 114.237 | 98.5516 | 102.149 | 107.532 |
| 10000 | 322.108 | 275.961 | 310.867 | 308.692 |

**Heap Sort Steps**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 8064 | 6370 | 7301 | 7196 |
| 200 | 18698 | 14945 | 16872 | 16841 |
| 300 | 30221 | 24715 | 27449 | 27498 |
| 400 | 42280 | 34982 | 39109 | 38752 |
| 500 | 54727 | 45619 | 50268 | 50372 |
| 1000 | 122062 | 103057 | 113835 | 113133 |
| 4000 | 592687 | 514168 | 553490 | 552735 |
| 10000 | 1652409 | 1449434 | 1547579 | 1547648 |

**C Approximation Heap Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances  Average |
| 100 | 17.5108 | 13.8323 | 15.8539 | 15.6259 |
| 200 | 17.6452 | 14.1035 | 15.922 | 15.8928 |
| 300 | 17.6614 | 14.4436 | 16.0414 | 16.07 |
| 400 | 17.6418 | 14.5966 | 16.3186 | 16.1697 |
| 500 | 17.6124 | 14.6812 | 16.1774 | 16.2108 |
| 1000 | 17.6703 | 14.919 | 16.4793 | 16.3777 |
| 4000 | 17.8648 | 15.4981 | 16.6833 | 16.6606 |
| 10000 | 17.9408 | 15.737 | 16.8026 | 16.8034 |

**Insertion Sort Running Time In Milliseconds**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 0.2295 | 5.2381 | 2.3583 | 3.72549 |
| 200 | 0.3208 | 22.2707 | 9.2919 | 13.8807 |
| 300 | 0.299 | 44.4023 | 28.1532 | 29.0095 |
| 400 | 0.4017 | 69.5943 | 44.6818 | 53.344 |
| 500 | 0.4986 | 110.2 | 68.7498 | 83.6949 |
| 1000 | 1.5906 | 458.628 | 225.628 | 329.323 |
| 4000 | 6.4903 | 8439.44 | 4412.94 | 5319.72 |
| 10000 | 14.1005 | 51244 | 30699.4 | 38942.9 |

**Insertion Sort Steps**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 599 | 20399 | 10039 | 10380 |
| 200 | 1199 | 80799 | 43219 | 40841 |
| 300 | 1799 | 181199 | 92555 | 91610 |
| 400 | 2399 | 321599 | 160691 | 161028 |
| 500 | 2999 | 501999 | 259711 | 251573 |
| 1000 | 5999 | 2003999 | 1000019 | 1003541 |
| 4000 | 23999 | 32015999 | 15846627 | 16051039 |
| 10000 | 59999 | 200039999 | 99309943 | 100074763 |

**C Approximation Insertion Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances  Average |
| 100 | 0 | 2 | 1 | 1 |
| 200 | 0 | 2 | 1 | 1 |
| 300 | 0 | 2 | 1 | 1 |
| 400 | 0 | 2 | 1 | 1 |
| 500 | 0 | 2 | 1 | 1 |
| 1000 | 0 | 2 | 1 | 1 |
| 4000 | 0 | 2 | 1 | 1 |
| 10000 | 0 | 2 | 1 | 1 |

**Merge Sort Running Time In Milliseconds**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 4.8864 | 3.6226 | 3.7307 | 4.48856 |
| 200 | 9.2835 | 11.6945 | 8.4183 | 9.79402 |
| 300 | 14.1179 | 12.3364 | 14.0176 | 15.2257 |
| 400 | 16.8485 | 16.2653 | 18.2677 | 21.0907 |
| 500 | 21.7617 | 20.9747 | 23.7114 | 30.1851 |
| 1000 | 43.7531 | 46.2174 | 48.6439 | 58.8891 |
| 4000 | 198.855 | 239.178 | 216.227 | 271.214 |
| 10000 | 533.399 | 612.249 | 711.775 | 760.396 |

**Merge Sort Steps**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances Average |
| 100 | 5774 | 5298 | 6225 | 6257 |
| 200 | 12958 | 11906 | 14202 | 14215 |
| 300 | 20662 | 18970 | 22766 | 22802 |
| 400 | 28726 | 26422 | 31873 | 31832 |
| 500 | 36518 | 34078 | 40912 | 40948 |
| 1000 | 80046 | 74666 | 90393 | 90365 |
| 4000 | 376214 | 350694 | 429529 | 429548 |
| 10000 | 1044102 | 961894 | 1190897 | 1190759 |

**C Approximation Merge Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size | Sorted | Reverse | Random Permutation | 50 Instances  Average |
| 100 | 12 | 11 | 13 | 13 |
| 200 | 12 | 11 | 13 | 13 |
| 300 | 12 | 11 | 13 | 13 |
| 400 | 12 | 11 | 13 | 13 |
| 500 | 12 | 11 | 13 | 13 |
| 1000 | 12 | 11 | 13 | 13 |
| 4000 | 12 | 11 | 13 | 13 |
| 10000 | 12 | 11 | 13 | 13 |

A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generated

**A screenshot of a computer

Description automatically generated**A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**