

|  |  |  |  |
| --- | --- | --- | --- |
| Course Code | : | DMT211 | |
| Course Name | : | Algorithm Analysis and Design | |
| Lecturer | : | Geetha Kanaparan | |
| Academic Session | : | 2021/09 | |
| Assessment Title | : | Assignment Question 1 | |
| Submission Due Date | : | 19/11/2021 | |
| Prepared by |  |  | |
| : | Student ID | Student Name |
|  | DMT2002036 | Yim Jing Xiang |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Date Received | : |  | |
|  |  |  | |
| Feedback from Lecturer:  Mark: | | | |

**Own Work Declaration**

I/We hereby understand my/our work would be checked for plagiarism or other misconduct, and the softcopy would be saved for future comparison(s).

I/We hereby confirm that all the references or sources of citations have been correctly listed or presented and I/we clearly understand the serious consequence caused by any intentional or unintentional misconduct.

This work is not made on any work of other students (past or present), and it has not been submitted to any other courses or institutions before.



Signature: Yim Jing Xiang

Date: 18/11/2021

**Question c**

//C++ Program to sort parcels with Quick Sort Algorithm

//Inputs: The number of parcel ids, n, to generate and sort

//Outputs: i. Time taken to perform the sorting

ii. All unsorted parcel ids and cost

iii. All sorted parcel ids and cost

struct parcel {int parcel\_ID; float cost;};

FUNCTION randomParcel(int n)

{

dynamically initialize an array with the size of n

set the seed of the randomizer seed to current time

for i -> 0 to n

array[i].parcel\_ID <- i + 1

assign a random cost to the array[i] with randomizer

i++

set the seed of the randomizer seed to current time

for i -> 0 to n

int hold\_two

parcel parcel\_hold

assign a random value mod n to variable hold\_two

swap two elements in the array

i++

return the randomized array

}

FUNCTION partition(parcel \*parcelArray, int head, int tail)

{

id\_hold <- parcelArray[tail].parcel\_ID

point <- head – 1

for i -> head to tail

if (parcelArray[i].parcel\_ID < id\_hold)

point++

swap parcelArray[i] with parcelArray[point]

i++

swap parcelArray[tail] with parcelArray[point+1]

return value of point + 1

}

FUNCTION quicksort(parcel \*parcelArray, int head, int tail)

{

if (head < tail)

pivot <- partition(parcelArray, head, tail)

quickSort(parcelArray, head, pivot - 1)

quicksort(parcelArray, pivot + 1, tail)

}

FUNCTION main()

{

menu\_status <- true

show\_unsort <- false

show\_sort <- false

getting input from user and assign it to n

while(menu\_status == true)

prompt to ask user to show/hide unsorted or sorted parcel list

parcel \*parcelArray = new parcel[n]

parcel \*parcelArrayCopy = new parcel[n]

parcelArray <- randomParcel(n)

copy parcelArray into parcelArrayCopy

assign current time to start

quicksort(parcelArray, 0, n-1)

assign current time to end

timeTaken <- end – start

print out timeTaken

if (show\_unsort == true)

print out the unsorted parcel list

if (show\_sort == true)

print out the sorted parcel list

}

**Question d**

|  |  |
| --- | --- |
| Sample Size (10 Trials) | Average Time Taken (Nanosecond) |
| 1000 | 0 |
| 2000 | 0 |
| 4000 | 0 |
| 5000 | 305830 |
| 8000 | 797900 |
| 10000 | 998810 |
| 16000 | 1598030 |
| 20000 | 2188090 |
| 32000 | 3686310 |
| 50000 | 5785640 |
| 64000 | 7380150 |
| 100000 | 12262980 |
| 128000 | 15851030 |
| 256000 | 33514210 |
| 512000 | 73703120 |
| 600000 | 84865980 |
| 700000 | 110803990 |
| 800000 | 130850120 |
| 900000 | 153166950 |
| 1000000 | 172524970 |
| 10000000 | 6500702850 |

i.

ii.

**Question e**

Based on the growth rate for various n value above, the performance of the Quicksort algorithm is actually decent for a sample size like 10 million. When the sample size reached 5000 and above, there is a small amount of time used to sort the array. This is affected by how are the ids are being randomized. In the best case, it might be 0 nanoseconds but in average case, it will always take some time to sort the array as they are iterating 5000 log (5000) times. As the sample space is getting larger, the gradient of the line is larger. This is because as the n is growing, n log n will get larger gradually. The time taken for the sorting will become longer as more elements swapping is required. I have used the last element as the pivot. Although using random pivot might help in increase the efficiency of the algorithm but there is also a situation that using random pivot always having the worst case. So, as we are taking the average time taken, it doesn’t really matter which pivot we used.

To perform better than the Quicksort algorithm in this case, I will suggest using Counting Sort. The average time complexity of Counting Sort is O(N+K), which N is the sample space and K is the different between the smallest and the largest element in the array. (Shah, 2021) Although this algorithm only works on integer array, it fits our case perfectly. For example, in the case of 1000000 parcels generated, by using Quicksort, it will iterate 6000000 times but if we use Counting Sort, it only required to iterate 1999999 times. which is a lot less than using Quicksort. This algorithm will really help in sorting the parcel ids especially with a large sample space.

**Reference**

Shah, S. (2021, June 29). *Time & Space Complexity of Counting Sort*. OpenGenus IQ: Computing Expertise & Legacy. Retrieved November 17, 2021, from https://iq.opengenus.org/time-and-space-complexity-of-counting-sort/.

**Appendix**

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <iomanip>

#include <algorithm>

#include <chrono>

using namespace std;

*// This is a custom datatype to save both the parcel ID and cost of the parcel*

struct parcel

{

    int parcel\_ID;

    float cost;

};

*// This is a function to generate parcels with random ID and cost*

parcel \*randomParcel(int *n*)

{

    parcel \*array = new parcel[*n*];

    srand(time(NULL));

*// Assign the ID by iteration first, and assign random cost*

    for (int i = 0; i < *n*; i++)

    {

        array[i].parcel\_ID = i + 1;

        array[i].cost = float(rand() % 1000 + 1) + (rand() % 100 / (float)100);

    }

    srand(time(NULL));

*// Then, we will swap the elements to make the elements in the array "randomized"*

    for (int i = 0; i < *n*; i++)

    {

        int hold\_two;

        parcel parcel\_hold;

        hold\_two = rand() % *n*;

        parcel\_hold = array[i];

        array[i] = array[hold\_two];

        array[hold\_two] = parcel\_hold;

    }

    return array;

}

*// Partition function that returns the index + 1 of the pivot*

int partition(parcel \**parcelArray*, int *head*, int *tail*)

{

    int id\_hold = *parcelArray*[*tail*].parcel\_ID;

    int point = *head* - 1;

*// Comparison is done here to swap the elements smaller than the pivot to index point*

    for (int i = *head*; i < *tail*; i++)

    {

        if (*parcelArray*[i].parcel\_ID < id\_hold)

        {

            point++;

            parcel parcelHold = *parcelArray*[i];

*parcelArray*[i] = *parcelArray*[point];

*parcelArray*[point] = parcelHold;

        }

    }

*// pivot element is swapped with the index of (last smaller element) + 1*

    parcel parcelHold = *parcelArray*[*tail*];

*parcelArray*[*tail*] = *parcelArray*[point + 1];

*parcelArray*[point + 1] = parcelHold;

    return point + 1;

}

*// QuickSort Function that do the recursive calls*

void quickSort(parcel \**parcelArray*, int *head*, int *tail*)

{

    if (*head* < *tail*)

    {

*// Getting the pivot*

        int pivot = partition(*parcelArray*, *head*, *tail*);

*// Recursive calls here*

        quickSort(*parcelArray*, *head*, pivot - 1);

        quickSort(*parcelArray*, pivot + 1, *tail*);

    }

}

int main()

{

    bool menu\_status = true, show\_unsort = false, show\_sort = false;

*// Prompt user to input the number of parcels*

    int n;

    cout << "Please enter the number of parcels.\n";

    cin >> n;

*// This is the menu that the user can choose either show/hide the unsorted/sorted parcel list*

    while (menu\_status)

    {

        int choice;

        if (!show\_unsort)

        {

            if (!show\_sort)

            {

                cout << "1. Show all the unsorted parcel\n";

                cout << "2. Show all the sorted parcel\n";

                cout << "3. Proceed\n";

                cout << "Your Choice = ";

                cin >> choice;

                cout << endl;

                switch (choice)

                {

                case 1:

                    show\_unsort = true;

                    break;

                case 2:

                    show\_sort = true;

                    break;

                case 3:

                    menu\_status = false;

                    break;

                }

            }

            else

            {

                cout << "1. Show all the unsorted parcel\n";

                cout << "2. Hide all the sorted parcel\n";

                cout << "3. Proceed\n";

                cout << "Your Choice = ";

                cin >> choice;

                cout << endl;

                switch (choice)

                {

                case 1:

                    show\_unsort = true;

                    break;

                case 2:

                    show\_sort = false;

                    break;

                case 3:

                    menu\_status = false;

                    break;

                }

            }

        }

        else

        {

            if (!show\_sort)

            {

                cout << "1. Hide all the unsorted parcel\n";

                cout << "2. Show all the sorted parcel\n";

                cout << "3. Proceed\n";

                cout << "Your Choice = ";

                cin >> choice;

                cout << endl;

                switch (choice)

                {

                case 1:

                    show\_unsort = false;

                    break;

                case 2:

                    show\_sort = true;

                    break;

                case 3:

                    menu\_status = false;

                    break;

                }

            }

            else

            {

                cout << "1. Hide all the unsorted parcel\n";

                cout << "2. Hide all the sorted parcel\n";

                cout << "3. Proceed\n";

                cout << "Your Choice = ";

                cin >> choice;

                cout << endl;

                switch (choice)

                {

                case 1:

                    show\_unsort = false;

                    break;

                case 2:

                    show\_sort = false;

                    break;

                case 3:

                    menu\_status = false;

                    break;

                }

            }

        }

    }

    parcel \*parcelArray = new parcel[n];

    parcel \*parcelArrayCopy = new parcel[n];

*// The randomized parcel array is assigned to parcelArray*

    parcelArray = randomParcel(n);

*// Another copy is made just in case the user wants to show the unsorted parcel list*

    for (int i = 0; i < n; i++)

    {

        parcelArrayCopy[i] = parcelArray[i];

    }

*// Start and end time is assigned here to record the time taken for the sorting*

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

    quickSort(parcelArray, 0, n - 1);

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

*// Different time units are used here*

    auto timeTaken\_second = chrono::duration\_cast<chrono::seconds>(end - start);

    auto timeTaken\_millisecond = chrono::duration\_cast<chrono::milliseconds>(end - start);

    auto timeTaken\_microsecond = chrono::duration\_cast<chrono::microseconds>(end - start);

    auto timeTaken\_nanosecond = chrono::duration\_cast<chrono::nanoseconds>(end - start);

    cout << "Time taken to perform the sorting = " << timeTaken\_second.count() << " seconds\n";

    cout << "\t\t\t\t  = " << timeTaken\_millisecond.count() << " milliseconds\n";

    cout << "\t\t\t\t  = " << timeTaken\_microsecond.count() << " microseconds\n";

    cout << "\t\t\t\t  = " << timeTaken\_nanosecond.count() << " nanoseconds\n\n";

*// If the user chose to show the unsorted parcel list, this will run*

    if (show\_unsort)

    {

        cout << "----------------------\n";

        cout << "|Unsorted Parcel List|\n";

        cout << "----------------------\n";

        for (int i = 0; i < n; i++)

        {

            cout << "Parcel ID = " << parcelArrayCopy[i].parcel\_ID << endl;

            cout << "Parcel Cost = RM" << setprecision(2) << fixed << parcelArrayCopy[i].cost << endl;

        }

        cout << endl;

    }

*// If the user chose to show the sorted parcel list, this will run*

    if (show\_sort)

    {

        cout << "--------------------\n";

        cout << "|Sorted Parcel List|\n";

        cout << "--------------------\n";

        for (int i = 0; i < n; i++)

        {

            cout << "Parcel ID = " << parcelArray[i].parcel\_ID << endl;

            cout << "Parcel Cost RM= " << setprecision(2) << fixed << parcelArray[i].cost << endl;

        }

        cout << endl;

    }

    return 0;

}