**FACULTY OF ELECTRICAL**

**AND ELECTRONIC ENGINEERING  
2020 – 2021  
-------\*-------  
  
Project: Data structure and algorithms**

**Topic: Merge sort**

**ECE391: Computer System Engineering**

**Instructor: Trương Quang Vinh**

**Students: Tô Đức Thành 1851015**

**Châu Nguyễn Thái Sơn 1751079**

**Lê Ngọc Thịnh 1751094**

**Nguyễn Minh Huy 1851029**

**Ho Chi Minh, 12/2020**

**Content**

[**I.** **Introduction** 1](#_Toc59103907)

[1. Intro the problem: sorting algorithm? 1](#_Toc59103908)

[2. Describe the objectives: 1](#_Toc59103909)

[**II.** **Algorithm** 1](#_Toc59103910)

[1. Describe algorithm 1](#_Toc59103911)

[2. The flow charts 2](#_Toc59103912)

[3. Explain algorithm 4](#_Toc59103913)

[**III.** **Program** 5](#_Toc59103914)

[1. Sub-functions: 11](#_Toc59103915)

[2. Function for graphical animation: User interface. 12](#_Toc59103916)

# **Introduction**

## Intro the problem: sorting algorithm?

Technically, sorting is just simply arranging data into more readable format => In order to be retrieved quickly for later uses (database, phone numbers, pages in books, ….).

Importance of sorting lies within how optimal it is in finding the required data.

## Describe the objectives:

There are many different techniques available for sorting including: counting, comb, cycle, heap, merge, ... But we’re focusing on the merge sorting technique today.

# **Algorithm**

## Describe algorithm

Merge sort is based on an algorithm design pattern called **divide-and-conquer**. The divide-and-conquer pattern consists of the following three steps:

* Divide: If the input size is smaller than a certain threshold (one or two elements), solve the problem directly using a straightforward method and return the solution obtained. Otherwise, divide the input data into two or more disjoint subsets.
* Recur: Recursively solve the subproblems associated with the subsets.
* Conquer: Take the solutions to the subproblems and “merge” them into a solution to the original problem.

To sort sequence S with n elements using the three divide -and- conquer steps, the merge-sort algorithm proceeds as above steps:

Divide:

If S has zero or one element, return S immediately; it is already sorted. Otherwise (S has at least two elements), remove all the elements from S and put them into two sequences, S1, and S2, each containing about half of the elements of S; that is, S1 contains the first [n/2] elements of S; and S2 contains the remaining [n/2] elements.

Recur:

Recursively sort sequences S1 and S2.

Conquer:

Put back the elements into S by merging the sort sequences S1and S2 into a sorted sequence.

## The flow charts

We represent an execution of the merge-sort algorithm by means of a binary tree T, called the merge-sort tree. Each node of T represents a recursive invocation of the merge-sort algorithm. We associate the sequence S that is processed by the invocation associated with q, with each node q of T. The children of node q are associated with the recursive calls that process the subsequences S1 and S2 of S.

In particular, since the size of the input sequence roughly halves at each recursive call merge-sort, the height of the merge-sort tree is about log n.

Begin

N = array.length

True

N <= 1

End

False

Middle = N/2;

leftLength = middle;

rightLength = N - middle;

Merge(array, left, right);

Left[index] =array[Index];

Index++;

Index < middle

True

MergeSort(right);

False

rightIndex = 0

Index = middle

False

True

MergeSort(left);

Index < N

Right[rightIndex]= array[Index]

rightIndex++;

Index++;

## Explain algorithm

Mergesort(S)

if (S.size() <= 1) then

Return;

S1.size() = S.size() /2; { divide S sequence into S1, S2 sequences}

S2.size() = S.size() - S1.size(); { divide S sequence into S1, S2 sequences}

i ← j ← 0 ;

For (i < S1.size(); i++)

S1.insertBack(S[i]); { copy ith element of S to end of S1}

For( j < S2.size(); j++);

S2.insertBack(S[S2.size() +j]); { copy ith element of S to end of S2}

Mergesort(S1, S1.size()); { recur mergesort function for left sequence}

Mergesort(S2, S2.size()); { recur mergesort function for right sequences}

Merge(S1, S2, S1.size(), S2.size()); { sort left and right sequences }

Mergesort(S, S1,S2);

i ← j ← 0

While (i < S1.size() and j < S2.size()) do

If (S1[i] > S2[j]) then

S.insertBack(S1[i]); { copy ith element of S1 to end of S}

i ← i+1;

Else

S.insertBack(S2[j]);

j ← j+1 ; { copy ith element of S2 to end of S}

While i < S1.size() do

S.insertBack(S1[i]); {copy the remaining elements of S1 to S}

i ← i+1;

While j < S2.size() do

S.insertBack(S2[j]); {copy the remaining elements of S2 to S}

j ← j+1 ;

# **Program**

// C++ program for Merge Sort

#include <iostream>

using namespace std;

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;

// Create temp arrays

int L[n1], R[n2];

// Copy data to temp arrays L[] and R[]

for (int i = 0; i < n1; i++)

L[i] = arr[l + i];

for (int j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

// Merge the temp arrays back into arr[l..r]

// Initial index of first subarray

int i = 0;

// Initial index of second subarray

int j = 0;

// Initial index of merged subarray

int k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

// Copy the remaining elements of

// L[], if there are any

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

// Copy the remaining elements of

// R[], if there are any

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

// l is for left index and r is

// right index of the sub-array

// of arr to be sorted \*/

void mergeSort(int arr[], int l, int r)

{

cout << "\n[Splitting]: ";

for (int i = l; i < r + 1; i++)

{

cout << arr[i] << " ";

}

cout << "\n";

if (l >= r)

{

return; //returns recursively

}

int m = (l + r - 1) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

cout << "[Merging]: ";

for (int i = l; i < r + 1; i++)

{

cout << arr[i] << " ";

}

cout << "\n";

}

// Function to print an array

void printArray(int A[], int size)

{

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

cout << A[i] << " ";

}

int main()

{

int arr[] = {54, 26, 73, 17, 77, 23, 45};

int arr\_size = sizeof(arr) / sizeof(arr[0]);

cout << "\nGiven array is \n";

printArray(arr, arr\_size);

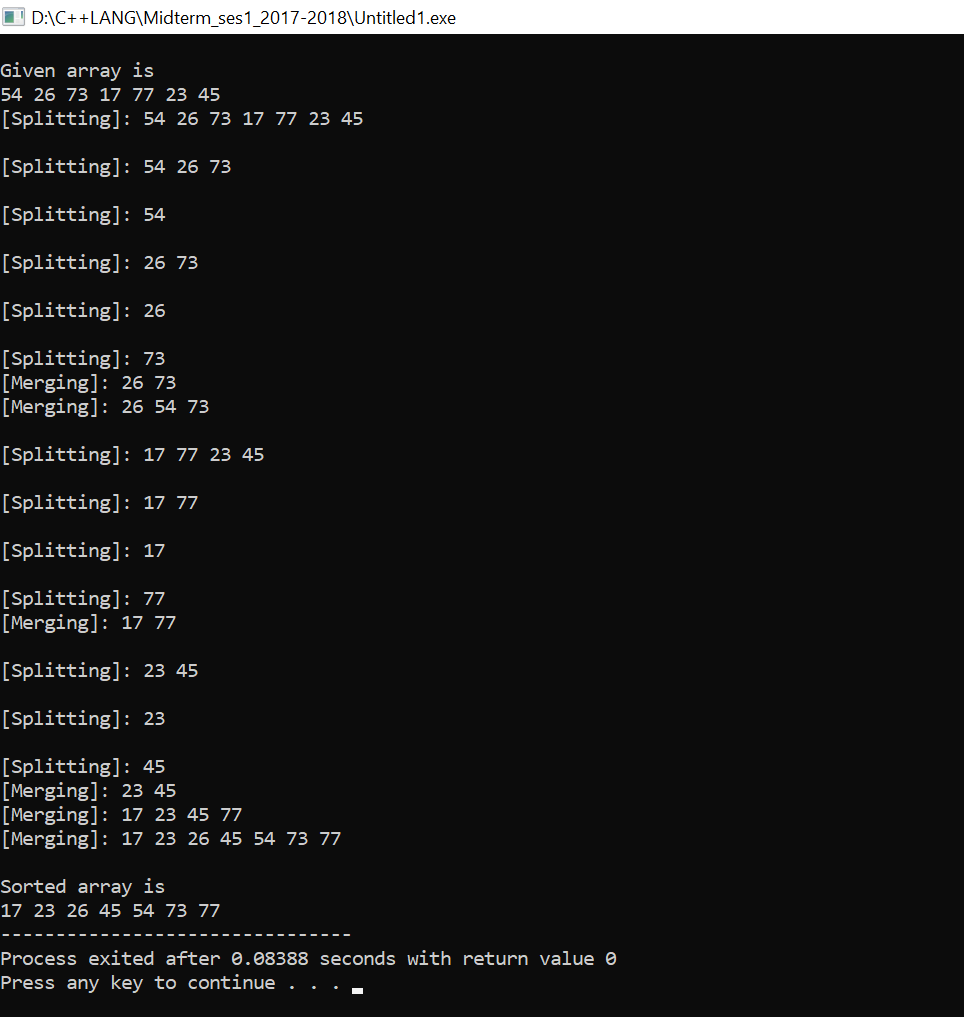
mergeSort(arr, 0, arr\_size - 1);

cout << "\nSorted array is \n";

printArray(arr, arr\_size);

return 0;

}



## Sub-functions:

void merge(int arr[], int l, int m, int r)

void mergeSort(int arr[], int l, int r)

## Function for graphical animation: User interface.

#include <SDL2/SDL.h>

#include <iostream>

#include <limits>

#include <time.h>

#include <string>

using namespace std;

const int SCREEN\_WIDTH = 910;

const int SCREEN\_HEIGHT = 750;

const int arrSize = 130;

const int rectSize = 7;

int arr[arrSize];

int Barr[arrSize];

SDL\_Window \*window = NULL;

SDL\_Renderer \*renderer = NULL;

bool complete = false;

bool init()

{

bool success = true;

if (SDL\_Init(SDL\_INIT\_VIDEO) < 0)

{

cout << "Couldn't initialize SDL. SDL\_Error: " << SDL\_GetError();

success = false;

}

else

{

if (!(SDL\_SetHint(SDL\_HINT\_RENDER\_SCALE\_QUALITY, "1")))

{

cout << "Warning: Linear Texture Filtering not enabled.\n";

}

window = SDL\_CreateWindow("Sorting Visualizer", SDL\_WINDOWPOS\_UNDEFINED, SDL\_WINDOWPOS\_UNDEFINED, SCREEN\_WIDTH, SCREEN\_HEIGHT, SDL\_WINDOW\_SHOWN);

if (window == NULL)

{

cout << "Couldn't create window. SDL\_Error: " << SDL\_GetError();

success = false;

}

else

{

renderer = SDL\_CreateRenderer(window, -1, SDL\_RENDERER\_ACCELERATED);

if (renderer == NULL)

{

cout << "Couldn't create renderer. SDL\_Error: " << SDL\_GetError();

success = false;

}

}

}

return success;

}

void close()

{

SDL\_DestroyRenderer(renderer);

renderer = NULL;

SDL\_DestroyWindow(window);

window = NULL;

SDL\_Quit();

}

void visualize(int x = -1, int y = -1, int z = -1)

{

SDL\_SetRenderDrawColor(renderer, 0, 0, 0, 0);

SDL\_RenderClear(renderer);

int j = 0;

for (int i = 0; i <= SCREEN\_WIDTH - rectSize; i += rectSize)

{

SDL\_PumpEvents();

SDL\_Rect rect = {i, 0, rectSize, arr[j]};

if (complete)

{

SDL\_SetRenderDrawColor(renderer, 100, 170, 110, 255);

SDL\_RenderDrawRect(renderer, &rect);

}

else if (j == x || j == z)

{

SDL\_SetRenderDrawColor(renderer, 100, 170, 110, 255);

SDL\_RenderFillRect(renderer, &rect);

}

else if (j == y)

{

SDL\_SetRenderDrawColor(renderer, 165, 105, 189, 255);

SDL\_RenderFillRect(renderer, &rect);

}

else

{

SDL\_SetRenderDrawColor(renderer, 170, 183, 184, 255);

SDL\_RenderDrawRect(renderer, &rect);

}

j++;

}

SDL\_RenderPresent(renderer);

}

void merge2SortedArrays(int a[], int start, int end)

{

int size\_output = (end - start) + 1;

int \*out = new int[size\_output];

int middle = (start + end) / 2;

int i = start, j = middle + 1, k = 0;

while (i <= middle && j <= end)

{

if (a[i] <= a[j])

{

out[k] = a[i];

visualize(i, j);

i++;

k++;

}

else

{

out[k] = a[j];

visualize(i, j);

j++;

k++;

}

}

while (i <= middle)

{

out[k] = a[i];

visualize(-1, i);

i++;

k++;

}

while (j <= end)

{

out[k] = a[j];

visualize(-1, j);

j++;

k++;

}

int x = 0;

for (int l = start; l <= end; l++)

{

a[l] = out[x];

visualize(l);

SDL\_Delay(20);

x++;

}

delete[] out;

}

void mergeSort(int a[], int start, int end)

{

if (start >= end)

{

return;

}

int middle = (start + end) / 2;

mergeSort(a, start, middle);

mergeSort(a, middle + 1, end);

merge2SortedArrays(a, start, end);

}

void loadArr()

{

memcpy(arr, Barr, sizeof(int) \* arrSize);

}

void randomizeAndSaveArray()

{

unsigned int seed = (unsigned)time(NULL);

srand(seed);

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

{

int random = rand() % (SCREEN\_HEIGHT);

Barr[i] = random;

}

}

void execute()

{

if (!init())

{

cout << "SDL Initialization Failed.\n";

}

else

{

randomizeAndSaveArray();

loadArr();

SDL\_Event e;

bool quit = false;

while (!quit)

{

while (SDL\_PollEvent(&e) != 0)

{

if (e.type == SDL\_QUIT)

{

quit = true;

complete = false;

}

else if (e.type == SDL\_KEYDOWN)

{

switch (e.key.keysym.sym)

{

case (SDLK\_q):

quit = true;

complete = false;

cout << "\nEXITING SORTING VISUALIZER.\n";

break;

case (SDLK\_0):

randomizeAndSaveArray();

complete = false;

loadArr();

cout << "\nNEW RANDOM LIST GENERATED.\n";

break;

case (SDLK\_1):

loadArr();

cout << "\nMERGE SORT STARTED.\n";

complete = false;

mergeSort(arr, 0, arrSize - 1);

complete = true;

cout << "\nMERGE SORT COMPLETE.\n";

break;

}

}

}

visualize();

}

close();

}

}

bool controls()

{

cout << "WARNING: Giving repetitive commands may cause latency and the visualizer may behave unexpectedly. Please give a new command only after the current command's execution is done.\n\n"

<< "Available Controls inside Sorting Visualizer:-\n"

<< " Use 0 to Generate a different randomized list.\n"

<< " Use 1 to start Merge Sort Algorithm.\n"

<< " Use q to exit out of Sorting Visualizer\n\n"

<< "PRESS ENTER TO START SORTING VISUALIZER...\n\n"

<< "Or type -1 and press ENTER to quit the program.";

string s;

getline(cin, s);

if (s == "-1")

{

return false;

}

return true;

}

void intro\_mess()

{

cout << "==============================Sorting Visualizer==============================\n\n"

<< "Visualization of Merge sort algorithm in C++ with SDL2 Library. A sorting algorithm is an algorithm that puts the elements of a list in a certain order. While there are a large number of sorting algorithms, in practical implementations a few algorithms predominate.\n"

<< "In this implementation of sorting visualizer, we'll be looking at Merge sort algorithms and visually comprehend its working.\n"

<< "The list size is fixed to 130 elements. You can randomize the list and select any type of sorting algorithm to call on the list from the given options. Here, all sorting algorithms will sort the elements in ascending order. The sorting time being visualized for an algorithm is not exactly same as their actual time complexities. The relatively faster algorithm like Merge Sort have been delayed so that they could be properly visualized.\n\n"

<< "Press ENTER to show controls...";

string s;

getline(cin, s);

if (s == "\n")

{

return;

}

}

int main(int argc, char \*args[])

{

intro\_mess();

char n;

while (1)

{

controls();

execute();

}

return 0;

}