//COS10007 – Developing Technical Software//  
//Lab 7 – Week 7//  
//Student ID – 103234103 – Son Nguyen//

**1/**  
**Let the array of number be [ 64, 25, 12 , 22 , 11 , 15]**

**Bubble sort: Bubble sort is the simplest sorting algorithm. By comparing the next elements to the original elements to compare which one is greater. It will continue to this swap until the condition is satisfy   
Eg. [64 , 25] would be the first and second elements – it would compare 64 and 25 , if 64 is larger than 25 – it’ll commit the swap otherwise it’ll leave the element where it is, then it’ll repeat after it has placed 64 in the correct location – repeat with 25.  
--------------------------------------------------------------------------------------------------------------------------------------  
Selection sort: Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from an unsorted portion of the list and move it to the sorted portion of the list  
Eg. With the set of number from above – the first position which is 64 will traverse the array and define that 11 is the lowest value – placing itself [11 , 25 , 12 , 22 , 64 , 15] , next number will repeat the procedure and so on till the array is sorted.  
--------------------------------------------------------------------------------------------------------------------------------------Insertion sort: Insertion sort is a simple sorting algorithm that works similar to how we place playing cards – the procedure work by splitting the array into 2 piles – one sorted and the other unsorted and it’ll placed the unsorted into the position corresponding to the number value in the sorted pile.  
Eg. Taking the first 3 number – [64, 25 , 12] ; the first number is then compared to the 2nd number then it is moved to an organised position then the same is re-done to create the sorted pile [12,25,64] then the next 3 number become unsorted which will be place slowly after comparison making the array [ 11 , 12 , 15 , 22 , 25 , 64]**

**--------------------------------------------------------------------------------------------------------------------------------------  
Merge sort: Merge is sort is breaking down the array to smaller subarray ; sorting them and then re-merge them into the array and then do a final sort  
Eg. Using the above array , they are divide into 3 sub-array [64,25] ; [12,22] ; [11,15]. They are then sorted [25,64] ; [12,22] ; [11,15] then they will merge and then re-sorted to complete the array.**

**--------------------------------------------------------------------------------------------------------------------------------------  
Quick sort : Quick sort is a divide and conqueror algorithm. It pick an elements as a pivot and partitions the given array around the picked pivot.  
Diagram** **2/**

**a/ Using bubble sort**#include <stdio.h>

#include <string.h>

struct student {

char name[10];

int rank;

};

int main() {

struct student arr[5] = {{"John", 3}, {"Alice", 1}, {"Bob", 5}, {"Emily", 2}, {"David", 4}};

// print initial array

printf("Initial array:\n");

for (int i = 0; i < 5; i++) {

printf("%s, %d\n", arr[i].name, arr[i].rank);

}

printf("\n");

// bubble sort

for (int i = 0; i < 4; i++) { // iterate n-1 times

for (int j = 0; j < 4-i; j++) { // iterate n-1-i times

if (arr[j].rank > arr[j+1].rank) {

// swap

struct student temp = arr[j];

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

arr[j+1] = temp;

}

}

}

// print sorted array

printf("Sorted array:\n");

for (int i = 0; i < 5; i++) {

printf("%s, %d\n", arr[i].name, arr[i].rank);

}

return 0;

}

**Output   
Text

Description automatically generated**

**b/ Using selection sort**#include <stdio.h>

#include <string.h>

struct student {

char name[10];

int rank;

};

// Function to perform selection sort on the student array based on their rank

void selectionSort(struct student arr[], int n) {

int i, j, min\_idx;

struct student temp;

// One by one move boundary of unsorted subarray

for (i = 0; i < n - 1; i++) {

// Find the minimum element in unsorted array

min\_idx = i;

for (j = i + 1; j < n; j++) {

if (arr[j].rank < arr[min\_idx].rank)

min\_idx = j;

}

// Swap the found minimum element with the first element

temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

int main() {

struct student students[5] = {

{"John", 2},

{"Alice", 5},

{"Bob", 1},

{"David", 4},

{"Mary", 3}

};

int n = sizeof(students) / sizeof(students[0]);

printf("Initial array:\n");

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

printf("%s - %d\n", students[i].name, students[i].rank);

}

// Sort the students array using selection sort

selectionSort(students, n);

printf("\nSorted array:\n");

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

printf("%s - %d\n", students[i].name, students[i].rank);

}

return 0;

}

**Output   
Text

Description automatically generated  
  
3/   
a/ Using Insertion sort**#include <stdio.h>

#include <string.h>

struct veg {

char item[10];

int price;

};

void insertionSort(struct veg arr[], int n) {

int i, j;

struct veg temp;

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

temp = arr[i];

j = i - 1;

while (j >= 0 && arr[j].price > temp.price) {

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

j--;

}

arr[j + 1] = temp;

}

}

void printArray(struct veg arr[], int n) {

int i;

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

printf("%s\t%d\n", arr[i].item, arr[i].price);

printf("\n");

}

int main() {

struct veg vegetables[5] = {{"tomato", 10},

{"carrot", 20},

{"potato", 15},

{"onion", 5},

{"pepper", 25}};

int n = sizeof(vegetables) / sizeof(vegetables[0]);

printf("Initial array:\n");

printArray(vegetables, n);

insertionSort(vegetables, n);

printf("Sorted array:\n");

printArray(vegetables, n);

return 0;

}

**Output   
A picture containing text

Description automatically generated  
b/ Using Merge sort**#include <stdio.h>  
#include <string.h>

struct veg {

char item[10];

int price;

};

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

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

struct veg L[n1], R[n2];

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

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

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

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

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i].price <= R[j].price) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(struct veg arr[], int l, int r) {

if (l < r) {

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

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(struct veg arr[], int n) {

int i;

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

printf("%s\t%d\n", arr[i].item, arr[i].price);

printf("\n");

}

int main() {

struct veg vegetables[5] = {{"tomato", 10},

{"carrot", 20},

{"potato", 15},

{"onion", 5},

{"pepper", 25}};

int n = sizeof(vegetables) / sizeof(vegetables[0]);

printf("Initial array:\n");

printArray(vegetables, n);

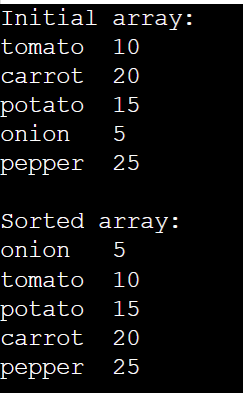
mergeSort(vegetables, 0, n - 1);

printf("Sorted array:\n");

printArray(vegetables, n);

return 0;

}

**Output   
  
c/ Using quick sort**#include <stdio.h>

#include <string.h>

struct veg {

char item[10];

int price;

};

void swap(struct veg\* a, struct veg\* b) {

struct veg t = \*a;

\*a = \*b;

\*b = t;

}

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

int pivot = arr[high].price;

int i = (low - 1);

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

if (arr[j].price <= pivot) {

i++;

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

}

}

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

return (i + 1);

}

void quickSort(struct veg arr[], int low, int high) {

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(struct veg arr[], int n) {

int i;

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

printf("%s\t%d\n", arr[i].item, arr[i].price);

printf("\n");

}

int main() {

struct veg vegetables[5] = {{"tomato", 10},

{"carrot", 20},

{"potato", 15},

{"onion", 5},

{"pepper", 25}};

int n = sizeof(vegetables) / sizeof(vegetables[0]);

printf("Initial array:\n");

printArray(vegetables, n);

quickSort(vegetables, 0, n - 1);

printf("Sorted array:\n");

printArray(vegetables, n);

return 0;

}

**Output -   
Text

Description automatically generated with low confidence**