Algorithms Assignment

Iria Parada
C22305863

TUDublin
Algorithms and Design and Problem

CMPU1001

Algorithms Assignment

Q1 –

I designed an array of structures “modules” that holds the code of the 4 modules, if they are full or part time, the admission numbers or maximum numbers, the current number of students and another structure that holds the name and surname of all the students.
With this structure you can manipulate and store this data, by entering the module you want to join/ leave and you will be registered into the array of structures. When displaying the structure, you can see all the data mentioned before, the code of the module, type, maximum/current number of students and their names/surnames. You can change the data every time you run the code, and also while running you can add more student into a module/modules and also remove them.
I designed this structure in C programming, used different functions to add/remove people from the structure and also to display the data. In the main function there is a menu in a do...while loop, in which you can chose your preferred option.
Here is the code in C:

```c
#include <stdio.h>
#include <string.h>

// size ofchars
#define SIZE 10
// number of modules
#define MOD 4
// number of students max 13+9+14+6=42
#define MAXST 42

// structure inside module structure to hold all names/surnames
struct names
{
    char firstname[SIZE];
    char surname[SIZE];
};
// structure to hold all data from each module
struct modules
{
    char code[SIZE];
    // 1 FULL -- 0 PART
    int type;
    int maximum;
    int current;
    struct names students[MAXST];
};
```

```c
// functions for each option
void join(struct modules *modsf);
void leave(struct modules *modsf);
void display(struct modules *modsf);

int main()
{
    // constant module codes, and max amount of students
    struct modules mods[MOD] = {{"DT265A", 0, 13, 0, {' ', ' '}},
                                {"DT265C", 0, 9, 0, {' ', ' '}},
                                {"DT265B", 1, 14, 0, {' ', ' '}},
                                {"DT8900", 1, 6, 0, {' ', ' '}}};

    // menu choice
    int choice;
    // end while loop
    int end = 1;

    do
    {
        // menu
        printf("1.Join module\n");
        printf("2.Leave module\n");
        printf("3.DIsplay modules\n");
        printf("4.Quit program\n");

        printf("\nEnter your choice:\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            join(mods);
            break;
        }
        case 2:
        {
            leave(mods);
            break;

        } // end case2
        case 3:
        {
            display(mods);
            break;
        } // end case3
```

```
        case 4:
        {
            end = -1;
            break;
        } // end 4
        // anyother number thats not in the menu
        default:
        {
            printf("\nPlease enter an option from the menu");
            break;
        }

        } // end switch

    } while (end == 1); // end while
    return 0;
} // end main

void join(struct modules *modsf)
{
    // modulechoice
    int modch = 0;
    // enter the module of your choice
    char mchoice[SIZE];
    int find = 0;
    char firstname[SIZE];
    char surname[SIZE];
    // enter what module to join
    printf("what module do you want to join:\n");
    scanf("%s", mchoice);
    // enter firstname and surname
    printf("Please enter your name:\n");
    scanf("%s", firstname);
    printf("Please enter your surname:\n");
    scanf("%s", surname);

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

        // compare/ check that it exists
        modch = strcmp(modsf[i].code, mchoice);
        if (modch == 0)
        {
            // module has been found
            find = 1;
```

```
            // if number of students is less than the max amount
            if (modsf[i].current < modsf[i].maximum)
            {
                // add one person to the module
                // register them
                modsf[i].current++;
                strcpy(modsf[i].students[modsf[i].current - 1].firstname,
firstname);
                strcpy(modsf[i].students[modsf[i].current - 1].surname,
surname);
                printf("\nYou have been added to the module\n");

            }    // end if
            else // number of students is greater than max
            {
                printf("\nModule is full\n");
            } // end else
            break;
        } // end if

    } // end for
    // if module is not found
    if (find == 0)
    {
        printf("Please chose an existing
module\nDT265A\nDT265C\nDT265B\nDT8900\n");
    } // end if
} // end function join

void leave(struct modules *modsf)
{
    // modulechoice
    int modch = 0;
    int nameexist = 0;
    char mchoice[SIZE];
    int find = 0;
    char surname[SIZE];
    // enter what module to leave
    printf("what module do you want to leave:\n");
    scanf("%s", mchoice);
    printf("Please enter your surname:\n");
    scanf("%s", surname);
    for (int i = 0; i < MOD; i++)
    {
        // compare/ check that it exists
        modch = strcmp(modsf[i].code, mchoice);
```

```
        if (modch == 0)
        {
            // module has been found
            find = 1;
            for (int j = 0; j < modsf[i].current; j++)
            {
                nameexist = strcmp(modsf[i].students[j].surname, surname);
                if (nameexist == 0)
                {
                    // SUBSTRACT one person to the module
                    // UNregister them
                    modsf[i].current--;
                    for (int k = 0; k < modsf[i].current; k++)
                    {
                        strcpy(modsf[i].students[k].surname,
modsf[i].students[k + 1].surname);
                    }

                    printf("\nyou have exited the module\n");
                    break;
                } // end if

            } // end for

        } // end if

    } // end for
    // if module not found
    if (find == 0)
    {
        printf("Please chose an existing
module\nDT265A\nDT265C\nDT265B\nDT8900");
    } // end if
} // end function leave

void display(struct modules *mods)
{
    // display all the data
    printf("\nFULL-TIME = 1\nPART-TIME = 0\n");
    printf("\nMODULE:  TYPE:  MAX:  CURRENT:\n ");
    for (int i = 0; i < MOD; i++)
    {
        printf("\n%s     %d    %d     %d ", mods[i].code, mods[i].type,
mods[i].maximum, mods[i].current);
        for (int j = 0; j < mods[i].current; j++)
        {
            printf(" %s %s \n", mods[i].students[j].firstname,
mods[i].students[j].surname);
```

```
        } // end for inner

    } // end for outer

} // end display
```

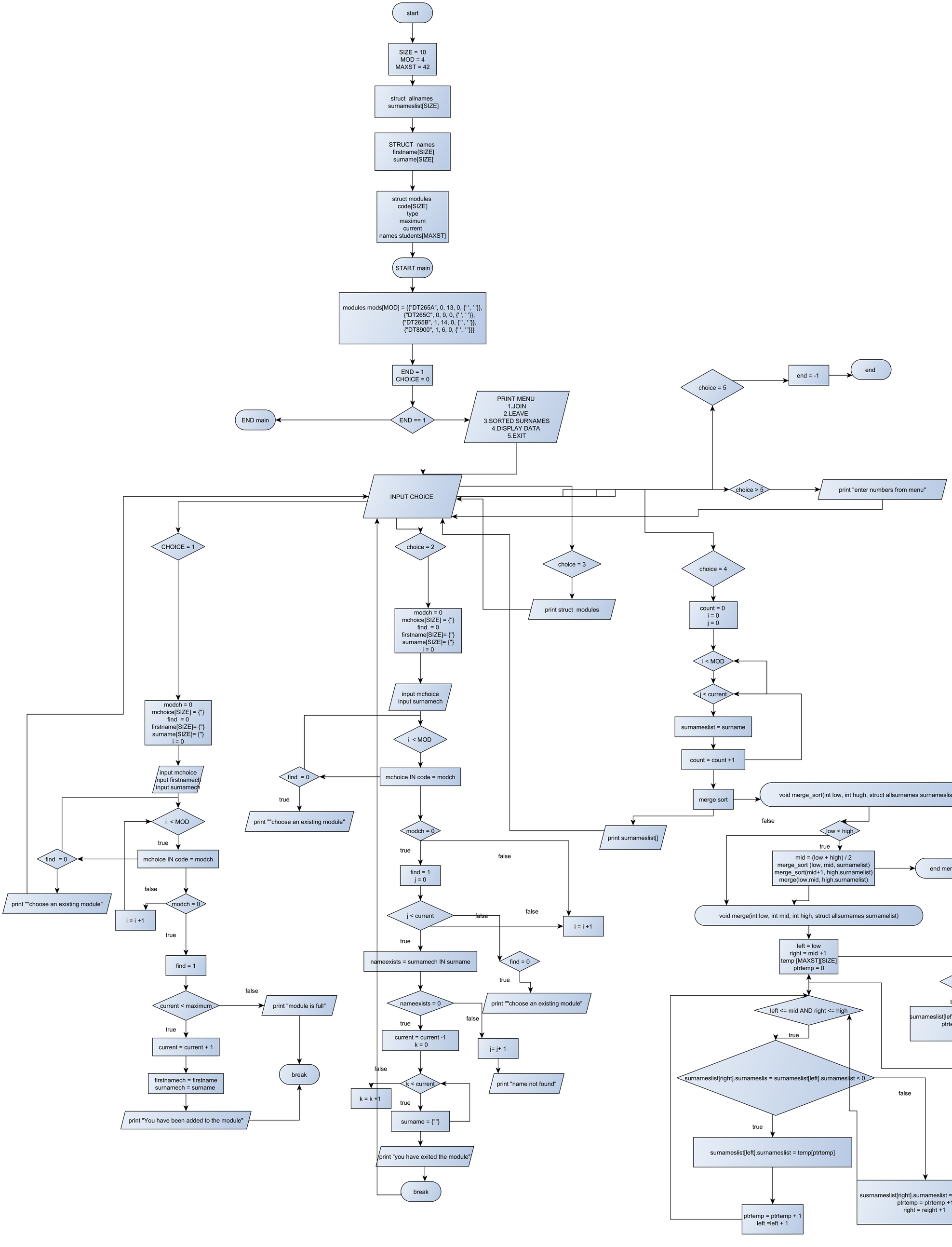
Algorithms Assignment

Q2 –

In a flowchart I have an algorithm that combines the four lists(array of structures that I made in the previous question) into one main list, I created another array of structures of the size of the maximum amount of students in all modules ”allsurnames”.Then I have sorted those surnames alphabetically, using the merge sort.

a) I used the merge sort to sort the surnames alphabetically, as it is a big amount of data 42 surnames maximum, the merge sort seemed the adequate as it sorts the data quicker, and accurately.

b) The big O of the merge sort is O(nLogn), it is more efficient it repeatedly divides the data by halves and then swaps the data in the correct order. It is the fastest sort algorithm in comparison to other studied in the module (bubble sort, insertion sort),it also works wwll in bigger data sizes such as this one.
The algorithm divides the list in two (left- right), then it assigns the index number for the low, middle and high position in the list. The big O, analyses the efficiency of the algorithm “n” is the variable that suggest the number of things.
The merge sort is in reality a combination of two algorithms.(O(n) x O(logn))
This sorting algorithms big O is is O(nLogn), because the divide and conquer part of the sort is O(log n) and the combination part is O(n) .


start
SIZE = 10
MOD = 4
MAXST = 42
struct allnames
surnameslist[SIZE]
STRUCT names
firstname[SIZE]
surname[SIZE[
struct modules
code[SIZE]
type
maximum
current
names students[MAXST]
START main
modules mods[MOD] = {{"DT265A", 0, 13, 0, {' ', ' '}},
{"DT265C", 0, 9, 0, {' ', ' '}},
{"DT265B", 1, 14, 0, {' ', ' '}},
{"DT8900", 1, 6, 0, {' ', ' '}}}
END = 1
CHOICE = 0
END == 1
END main
PRINT MENU
1.JOIN
2.LEAVE
3.SORTED SURNAMES
4.DISPLAY DATA
5.EXIT
INPUT CHOICE
choice = 5
end = -1
end
choice > 5
print "enter numbers from menu"
CHOICE = 1
modch = 0
mchoice[SIZE] = {''}
find = 0
firstname[SIZE]= {''}
surname[SIZE]= {''}
i = 0
input mchoice
input firstnamech
input surnamech
i < MOD
true
find = 0
mchoice IN code = modch
print ""choose an existing module"
false
modch = 0
i = i +1
true
find = 1
current < maximum
false
print "module is full"
true
current = current + 1
firstnamech = firstname
surnamech = surname
print "You have been added to the module"
break
choice = 2
modch = 0
mchoice[SIZE] = {''}
find = 0
firstname[SIZE]= {''}
surname[SIZE]= {''}
i = 0
input mchoice
input surnamech
i < MOD
find = 0
true
print ""choose an existing module"
mchoice IN code = modch
modch = 0
true
find = 1
j = 0
j < current
false
i = i +1
true
nameexists = surnamech IN surname
nameexists = 0
true
current = current -1
k = 0
k < current
false
k = k +1
true
surname = {""}
print "you have exited the module"
break
false
find = 0
true
print ""choose an existing module"
false
j= j+ 1
print "name not found"
choice = 3
print struct modules
print surnameslist[]
choice = 4
count = 0
i = 0
j = 0
i < MOD
j < current
surnameslist = surname
count = count +1
merge sort
void merge_sort(int low, int hugh, struct allsurnames surnameslists)
false
low < high
true
mid = (low + high) / 2
merge_sort (low, mid, surnamelist)
merge_sort(mid+1, high,surnamelist)
merge(low,mid, high,surnamelist)
end merge_sort
void merge(int low, int mid, int high, struct allsurnames surnamelist)
left = low
right = mid +1
temp [MAXST][SIZE]
ptrtemp = 0
left < = mid
true
surnameslist[left].surnamelist = temp[ptrtemp]
ptrtemp = ptrtemp + 1
left = left + 1
false
left <= mid AND right <= high
true
surnameslist[right].surnameslis = surnameslist[left].surnameslist < 0
false
true
surnameslist[left].surnameslist = temp[ptrtemp]
ptrtemp = ptrtemp + 1
left =left + 1
susrnameslist[right].surnameslist = temp[ptrtemp]
ptrtemp = ptrtemp +1
right = reight +1
right <= high
false
true
surnameslist[right].surnamelist = temp[ptrtemp]
ptrtemp = ptrtemp + 1
right = right + 1
i = 0
i < ptrtemp
temp[i] = surnamelslist[low +i].surnamelist
i = i +1

Algorithms Assignment

Q3 –

a) I used the linear search algorithm to search on the main list for all full-time students.
b) The big O of the linear search is O(n). As it goes through all the data individually. It searches for the full time modules in this case "1" (part time is 0), it searches I tin the structure in the array of "type" once it has found it prints the surname of the student enrolled in the first index, then continuous to the second number of the array and so on. Linear search is not very effective as it is very slow, it goes through all the data individually.

```
Start program

SIZE = 10
MOD = 4
MAXST = 42

struct all surnames
{
   char surnameslist[SIZE]
}
struct names
{
   char firstname[SIZE]
   char surname[SIZE]
}
struct modules
{
   char code[SIZE]
   int type
   int maximum
   int current
   struct names students[MAXST]
}

start main

   struct modules mods[MOD]= {{"DT265A", 0, 13, 0, {' ', ' '}},
                              {"DT265C", 0, 9, 0, {' ', ' '}},
                              {"DT265B", 1, 14, 0, {' ', ' '}},
                              {"DT8900", 1, 6, 0, {' ', ' '}}}
   struct allsurnames surnames[MAXST];
   int choice;
   int end = 1;
   char searchkey[SIZE];

   DO

      print {Menu 1.join 2.leave 3.Sorted Surnames 4.Display data 5.FULL-TIME students 6.what is
your module 7.exit }
```

```
    input choice

    switch (choice)
      case 1
        join(mods)
        break
      case 2
        leave(mods)
        break
      case 3
        sorted_surnames(surnames,mods)
        break
      case 4
        display(mods)
        break
      case 5
        PRINT "FULL-TIME students"
        linear_search(mods)
        break
      case 6
        PRINT "enter you surname"
        INPUT searchkey

        int find = -1
        int result

        FOR i = 0 , i < MOD , i = i +1
          IF (mods[i].current > 0 )
            result = binary_search(mods[i].students,mods[i].current, searchkey)
            IF (result NOT -1)
              find = i
              break
            END IF
          END IF
        END FOR
        IF (find NOT -1)
          PRINT "Name has been found"
        END IF
        ELSE
          PRINT "studentnot found"
        END ELSE
        break
      case 7
        end = -1
        break
      default
        PRINT "choose number form the menu"
        break
    END SWITCH
  END DO WHILE (end == 1 )
END MAIN
```

```
START join (struct modules POINTER modsf)
  int modch = 0
   char mchoice[SIZE]
   int find = 0
   char firstname[SIZE]
   char surname[SIZE]

   INPUT mchoice, firstname, surname

   FOR i= 0, i < MOD ,i= i+1
      modch = compare(modsf[i].code , mchoice)
      IF (modch == 0)
         find = 1
         IF (modsf[i].current < modsf[i].maximum)
            modsf[i].current = modsf[i].current + 1
            COPY (firstname INTO modsf[i].students[modsf[i].current - 1].firstname)
            COPY (surname INTO modsf[i].students[modsf[i].current - 1].surname)
            PRINT "you have been added to the module"
         END IF
         ELSE
            PRINT "module is full"
         END ELSE
         break
      END IF
   END FOR

   IF (find == 0 )
      PRINT "choose an exsisting module"
   END IF
END FUNCTION JOIN

START leave(struct modules POINTER modsf)
   int modch = 0
   int namexist = 0
   char mchoice[SIZE]
   int find = 0
   char surname[SIZE]

   INPUT mchoice,surname
   FOR (i= 0 , i < MOD, i = i +1)
      modch = COMPARE (mchoice TO modsf[i].code)
      IF (modch = 0)
         find = 1
         FOR j=0, j <modsf[i].current, j = j+1
            nameexist = COMPARE(surname TO modsf[i].students[j].surname)
            IF (nameexist = 0 )

               modsf[i].current = modsf[i].current - 1
               FOR k = 0 , k < modsf[i].current, k = k +1
                  COPY (modsf[i].students[k + 1].surname INTO modsf[i].students[k].surname )
```

```
            END FOR
            PRINT "you have exited the module"
            BREAK
          END IF
        END FOR
      END IF
    END FOR
    IF find = 0
      PRINT "please chose an existing module"
    END IF
END FUNCTION leave

START display (struct modules  POINTER mods)
    FOR i = 0, i < MOD , i = i +1
      PRINT "mods[i].code,mods[i].type,mods[i].maximum,mods[i].current"

      FOR j= 0 , j < mods[i].current, j = j +1
        PRINT mods[i].students[j].firstname,mods[i].students[j].surname
      END FOR
    END FOR
END DISPLAY FUNCTION

START linear_search(struct modules POINTER modsf)
    int key = 1
    found = 0
    FOR i =0, i < Mod , i = i +1
      FOR  j=0, j <MAXST, j = j +1
        IF (modsf[i].type == key)
          PRINT "modsf[i].students[j].firstname,modsf[i].students[j].surname"
          found = 1
          BREAK
        END IF
      END FOR
      IF found = 0
        PRINT "NO students in full time modules"
      END IF
    END FOR
END FUNCTION
```

Algorithms Assignment

Q4 –

O(log(N)) Is the big O for binary search, I am using this algorithm to search in the main list for a specific student by surname. Binary search is O(log(N)) because it relies on divide and conquer strategy to fins a specific value in the main list. Similarly to the mergesort the data is divided in low, middle and high positions. Using a search key (the surname) once the search key becomes the middle element it is returned(the index is returned), if search key is on the left subarray the high index of the middle element is subtracted by 1 if it is in the right subarray the lowest index of the middle element is added 1, and keeps doing this recursively until the search key is the middle element.

Start program

```
SIZE = 10
MOD = 4
MAXST = 42

struct all surnames
{
   char surnameslist[SIZE]
}
struct names
{
   char firstname[SIZE]
   char surname[SIZE]
}
struct modules
{
   char code[SIZE]
   int type
   int maximum
   int current
   struct names students[MAXST]
}

start main

   struct modules mods[MOD]= {{"DT265A", 0, 13, 0, {' ', ' '}},
                              {"DT265C", 0, 9, 0, {' ', ' '}},
                              {"DT265B", 1, 14, 0, {' ', ' '}},
                              {"DT8900", 1, 6, 0, {' ', ' '}}}
   struct allsurnames surnames[MAXST];
   int choice;
   int end = 1;
   char searchkey[SIZE];
```

```
DO

    print {Menu 1.join 2.leave 3.Sorted Surnames 4.Display data 5.FULL-TIME students 6.what is
your module 7.exit }

    input choice

    switch (choice)
      case 1
        join(mods)
        break
      case 2
        leave(mods)
        break
      case 3
        sorted_surnames(surnames,mods)
        break
      case 4
        display(mods)
        break
      case 5
        PRINT "FULL-TIME students"
        linear_search(mods)
        break
      case 6
        PRINT "enter you surname"
        INPUT searchkey

        int find = -1
        int result

        FOR i = 0 , i < MOD , i = i +1
          IF (mods[i].current > 0 )
            result = binary_search(mods[i].students,mods[i].current, searchkey)
            IF (result NOT -1)
              find = i
              break
            END IF
          END IF
        END FOR
        IF (find NOT -1)
          PRINT "Name has been found"
        END IF
        ELSE
          PRINT "studentnot found"
        END ELSE
        break
      case 7
        end = -1
        break
      default
```

```
            PRINT "choose number form the menu"
            break
        END SWITCH
    END DO WHILE (end == 1 )
END MAIN

START join (struct modules POINTER modsf)
   int modch = 0
    char mchoice[SIZE]
    int find = 0
    char firstname[SIZE]
    char surname[SIZE]

    INPUT mchoice, firstname, surname

    FOR i= 0, i < MOD ,i= i+1
        modch = compare(modsf[i].code , mchoice)
        IF (modch == 0)
            find = 1
            IF (modsf[i].current < modsf[i].maximum)
                modsf[i].current = modsf[i].current + 1
                COPY (firstname INTO modsf[i].students[modsf[i].current - 1].firstname)
                COPY (surname INTO modsf[i].students[modsf[i].current - 1].surname)
                PRINT "you have been added to the module"
            END IF
            ELSE
                PRINT "module is full"
            END ELSE
            break
        END IF
    END FOR

    IF (find == 0 )
        PRINT "choose an exsisting module"
    END IF
END FUNCTION JOIN

START leave(struct modules POINTER modsf)
    int modch = 0
    int namexist = 0
    char mchoice[SIZE]
    int find = 0
    char surname[SIZE]

    INPUT mchoice,surname
    FOR (i= 0 , i < MOD, i = i +1)
        modch = COMPARE (mchoice TO modsf[i].code)
        IF (modch = 0)
            find = 1
            FOR j=0, j <modsf[i].current, j = j+1
                nameexist = COMPARE(surname TO modsf[i].students[j].surname)
```

```
            IF (nameexist = 0 )

               modsf[i].current = modsf[i].current - 1
               FOR k = 0 , k < modsf[i].current, k = k +1
                  COPY (modsf[i].students[k + 1].surname INTO modsf[i].students[k].surname )
               END FOR
               PRINT "you have exited the module"
               BREAK
            END IF
         END FOR
      END IF
   END FOR
   IF find = 0
      PRINT "please chose an existing module"
   END IF
END FUNCTION leave

START binary_search(struct names student[], int n, char searchkey[])
   int low= 0
   int high = n -1
   int middle

   WHILE (low <= high)
      middle=(low + high)/2;

      int cmp = COMPARE (searchkey, student[middle].surname )

      IF (cmp == 0)
         RETURN middle
      END IF
      ELSE IF (cmp < 0)
         high = middle + 1
      END ELSE IF
      ELSE IF (cmp > 0)
         low = middle - 1
      END ELSE IF
   END WHILE
   RETURN -1
END FUNCTION
```

```c
#include <stdio.h>
#include <string.h>

// size ofchars
#define SIZE 10
// number of modules
#define MOD 4
// number of students max 13+9+14+6=42
#define MAXST 42

//one list for all surnames
struct allsurnames
{
    char surnameslist[SIZE];
};
//all names/surnames in all modules
struct names
{
    char firstname[SIZE];
    char surname[SIZE];
};
//structure for all 4 modules
struct modules
{
    char code[SIZE];
    // 1 FULL -- 0 PART
    int type;
    int maximum;
    int current;
    struct names students[MAXST];
};



void join(struct modules *modsf);
void leave(struct modules *modsf);
void display(struct modules *modsf);

void sorted_surnames( struct allsurnames *surnameslist,struct modules *modsf);
void merge_sort(int low, int high, struct allsurnames *surnameslist);
void merge(int low, int mid, int high, struct allsurnames *surnameslist);

void linear_search(struct modules *modsf);

int binary_search(struct names student[], int n, char searchkey[]);

int main()
{
```

```c
    struct modules mods[MOD] = {{"DT265A", 0, 13, 0, {' ', ' '}},
                                {"DT265C", 0, 9, 0, {' ', ' '}},
                                {"DT265B", 1, 14, 0, {' ', ' '}},
                                {"DT8900", 1, 6, 0, {' ', ' '}}};
    struct allsurnames surnames[MAXST];
    // menu choice
    int choice;
    // end while loop
    int end = 1;
    char searchkey[SIZE];

    do
    {
        // menu
        printf("\n\n__________\n");
        printf("1.FULL-TIME students\n");
        printf("2.Join module\n");
        printf("3.Leave module\n");
        printf("4.Display modules data\n");
        printf("5.Sorted surnames\n");
        printf("6.Whats your module\n");
        printf("7.Quit\n");

        printf("\nEnter your choice:\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            //tells you what students are enrolled in a fulltime course
            printf("\nFull-time students:\n");
            linear_search(mods);

            break;
        }
        case 2:
        {
            //add students to the modules
            join(mods);
            break;

        } // end case2
        case 3:
        {
            //remove students from the modules
            leave(mods);
            break;
```

```
        } // end case3
        case 4:
        {
            //display modules data
            display(mods);
            break;
        } // end 4
        case 5:
        {
            //all students in one list in alphabetical order
            sorted_surnames(surnames,mods);
            break;
        }
        case 6:
        {
            //tells you what module youare enrolled in based on your surname
            printf("\nEnter your Surname:");
            scanf("%s", searchkey);
            int find = -1;
            int result;
            for (int i = 0; i < MOD; i++)
            {
                if (mods[i].current >0)
                {
                    result
=binary_search(mods[i].students,mods[i].current,searchkey);
                    if(result !=-1)
                    {
                        find = i;
                        break;
                    }//end inner if
                }//end outer if
            }//end for


            if( find != -1)
            {
                printf("\nYour name has been found\n");
                printf("\n%s enrolled in %s\n",searchkey,mods[find].code);
            }//end if
            else
            {
                printf("\nStudent not found\n");
            }
            break;
        } // end case5
        case 7:
        {
```

```
            end = -1;
            break;
        } // end case6
        default:
        {
            printf("\nPlease enter an option from the menu\n");
            break;
        }

        } // end switch

    } while (end == 1); // end while
    return 0;
} // end main

void join(struct modules *modsf)
{
    // modulechoice
    int modch = 0;
    char mchoice[SIZE];
    int find = 0;
    char firstname[SIZE];
    char surname[SIZE];
    // enter what module to join
    printf("what module do you want to join:\n");
    scanf("%s", mchoice);
    printf("Please enter your name:\n");
    scanf("%s", firstname);
    printf("Please enter your surname:\n");
    scanf("%s", surname);

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

        // compare/ check that it exists
        modch = strcmp(modsf[i].code, mchoice);
        if (modch == 0)
        {
            // module has been found
            find = 1;
            // if number of students is less than the max amount
            if (modsf[i].current < modsf[i].maximum)
            {
                // add one person to the module
                // register them
                modsf[i].current++;
                strcpy(modsf[i].students[modsf[i].current - 1].firstname,
firstname);
```

```
                strcpy(modsf[i].students[modsf[i].current - 1].surname,
surname);
                printf("\nYou have been added to the module\n");

            }    // end if
            else // number of students is greater than max
            {
                printf("\nModule is full\n");
            } // end else
            break;
        } // end if

    } // end for
    if (find == 0)
    {
        printf("\nPlease chose an existing
module\nDT265A\nDT265C\nDT265B\nDT8900\n");
    } // end if
} // end function join

void leave(struct modules *modsf)
{
    // modulechoice
    int modch = 0;
    int nameexist = 0;
    char mchoice[SIZE];
    int find = 0;
    char surname[SIZE];
    // enter what module to join
    printf("\nwhat module do you want to leave:\n");
    scanf("%s", mchoice);
    printf("\nPlease enter your surname:\n");
    scanf("%s", surname);
    for (int i = 0; i < MOD; i++)
    {
        // compare/ check that it exists
        modch = strcmp(modsf[i].code, mchoice);

        if (modch == 0)
        {
            // module has been found
            find = 1;
            for (int j = 0; j < modsf[i].current; j++)
            {
                nameexist = strcmp(modsf[i].students[j].surname, surname);
                if (nameexist == 0)
                {
                    // SUBSTRACT one person to the module
```

```
                    // UNregister them
                    modsf[i].current--;
                    for (int k = 0; k < modsf[i].current; k++)
                    {
                        strcpy(modsf[i].students[k].surname,
modsf[i].students[k + 1].surname);
                    }

                    printf("\nYou have exited the module\n");
                    break;
                } // end if

            } // end for

        } // end if

    } // end for
    if (find == 0)
    {
        printf("\nPlease chose an existing
module\nDT265A\nDT265C\nDT265B\nDT8900");
    } // end if
} // end function leave

void display(struct modules *mods)
{
    printf("\nFULL-TIME = 1\nPART-TIME = 0\n");
    printf("\nMODULE:  TYPE:  MAX:  CURRENT:\n ");
    for (int  i = 0; i < MOD; i++)
   {
        printf("\n%s     %d    %d     %d
",mods[i].code,mods[i].type,mods[i].maximum,mods[i].current);
        for (int j = 0; j < mods[i].current; j++)
        {
            printf(" %s %s \n",
mods[i].students[j].firstname,mods[i].students[j].surname);
        }//end for inner

   }//end fo outer


} // end display
void sorted_surnames(struct allsurnames *surnameslist, struct modules *modsf)
{
    int count = 0;

    for (int i = 0; i < MOD; i++)
    {
```

```
        for (int j = 0; j < modsf[i].current; j++)
        {
            //copy all surnames from modules structure to new structure with
only surnames
            strcpy(surnameslist[count].surnameslist,
modsf[i].students[j].surname);
            count++;
        }
    }

    //call merge sort with sorted surnames
    merge_sort(0, count - 1, surnameslist);

    //display sorted surnames
    printf("Sorted surnames:\n");
    for (int i = 0; i < count; i++)
    {
        printf("%s\n", surnameslist[i].surnameslist);
    }
}

void merge_sort(int low, int high, struct allsurnames *surnameslist)
{

    if(low < high)
    {
        //divide the list in low, mid and high positions
        int mid = (low + high) / 2;
        merge_sort(low, mid, surnameslist);
        merge_sort(mid+1, high, surnameslist);
        //merge function
        merge(low, mid, high, surnameslist);
    }//end if
}//end merge_sort

void merge(int low, int mid, int high, struct allsurnames *surnameslist)
{
    int left = low;
    int right = mid + 1;
    char temp[MAXST][SIZE];
    int ptrtemp = 0;

    while(left <= mid && right <= high)
    {
        //compare surnames left and right and swap
        if(strcmp(surnameslist[left].surnameslist,
surnameslist[right].surnameslist) < 0)
        {
```

```
            //swap if condition true
            //use ptertemp as temporary variable to store surname and excecute
the swap
            //increment indeces(go to the next name)
            strcpy(temp[ptrtemp], surnameslist[left].surnameslist);
            ptrtemp++;
            left++;
        } //enf if
        else
        {
            //copy surname to temporary variable
            //increment indeces(go to the next name)
            strcpy(temp[ptrtemp], surnameslist[right].surnameslist);
            ptrtemp++;
            right++;
        }//end else
    }//end while

    while(left <= mid)
    {
        //copy surname to temporary variable
        //increment indeces(go to the next name)
        strcpy(temp[ptrtemp], surnameslist[left].surnameslist);
        ptrtemp++;
        left++;
    }//end while

    while(right <= high)
    {
        //copy surname to temporary variable
        //increment indeces(go to the next name)
        strcpy(temp[ptrtemp], surnameslist[right].surnameslist);
        ptrtemp++;
        right++;
    }//end while

    for(int i = 0; i < ptrtemp; i++)
    {
        //copy the temp variable into structure with only surnames
        //structre now holds the sorted surnames
        strcpy(surnameslist[low + i].surnameslist, temp[i]);
    }//end for
}//end merge

void linear_search(struct modules *modsf)
{
    //full time is represented by 1 in the structure
    int key = 1;
```

```
    int found = 0;

    //outer loop go trough  4 modules
    for (int i = 0; i < MOD ; i++)
    {
        //inner loop go trough students
        for (int j = 0; j < MAXST; j++)
        {
            //find 1 (full-time)
            if (modsf[i].type == key)
            {
                //display students in full-time
                printf("%s %s\n",
modsf[i].students[j].firstname,modsf[i].students[j].surname);
                found = 1;
                break;
            }//end if

        }//end for inner
        if (found = 0)
        {
            printf("\nNo students in FULL-TIME modules\n");
        }
    }//end for outer

}//end linear search

int binary_search(struct names student[], int n, char searchkey[])
{
    //divide data in positions low, mid,high
    int low =0;
    int high= n-1;
    int middle;

    while (low <= high)
    {
        middle =(low +high)/2;

        int cmp = strcmp(student[middle].surname,searchkey);

        if (cmp == 0)
        {
            //searck key will be found (returned) when is equals to the middle
element.
            return middle;

        }//end if
        else if (cmp < 0)
```

```
        {
            high = middle +1;

        }//end else if
        else if (cmp > 0)
        {
            low = middle -1 ;

        }//end else if
    }
    return -1;
}//end binary
```