CS 419 Compiler

Project Form

Project Idea: 3

Team NO: 6

Assigned Team Code: CO1903

ID	Name	Level& Department	Section(Day-from-to)	Role	Grade
20180329	عبدالغفار ناصر عبدالغفار ابوالعطا	Level 3/CS	Wednesday(4-6)	Team leader	
20180024	احمد إيهاب محمد محمد مصطفي	Level 3/CS	Wednesday(4-6)	Member	
20180331	عبدالله إسماعيل محمد محمد	Level 3/CS	Wednesday(4-6)	Member	
20180027	احمد جلال عبدالحميد السيد	Level 3/CS	Wednesday(4-6)	Member	
20180292	شهاب شريف محمد	Level 3/CS	Wednesday(2-4)	Member	
20180549	محمود احمد رجب سيد	Level 3/CS	Wednesday(4-6)	Member (Didn't participate in anything)	

C Code

```
#include <stdio.h>
#include <stdlib.h>

// Merge the two half into a sorted data

//mergeFunctionAlgorithm
void merge(char arr[], int left, int middle, int right)
{
    int i, j, k;
    int len1 = middle - left + 1;
    int len2 =  right - middle;

    // Create temp arrays
    char L[len1], R[len2];

    // Copy data to temp arrays L[] and R[]
    for (i = 0; i < len1; i++)
        L[i] = arr[left + i];

    for (j = 0; j < len2; j++)
        R[j] = arr[middle + 1 + j];

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

     i = 0; // Initial index of first sub array
     j = 0; // Initial index of second sub array
     k = left; // Initial index of merged sub array or it can be 0 also

     while (i < len1 && j < len2)
     {
         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 < len1)
     {
         arr[k] = L[i];
         i++;
         k++;
     }

     // Copy the remaining elements of R[], if there are any
     while (j < len2)
     {
         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

//mergeSortAlgorithm
void mergeSort(char arr[], int left, int right)
{
    if (left < right)
    {
        // Same as (l+r)/2, but avoids overflow for large l and h
        int middle = left + (right - left)/2;

        // Sort first and second halves
        mergeSort(arr, left, middle);
        mergeSort(arr, middle + 1, right);

        merge(arr, left, middle, right);
     }
 }

//SelectionSortAlgorithm
void swap_n(int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}

void swap_c(char *x, char *y){
    char temp = *x;
    *x = *y;
    *y = temp;
```

```
}

void MIN_n(int i, int arr[], int SIZE){
    int MIN;
    for(int j = i +1; j < SIZE; j++){
        MIN = i;
        if(arr[j] < arr[MIN]){
            MIN = j;
            swap_n(&arr[MIN], &arr[i]);
        }
    }
}

void MIN_c(int i, char arr[], int SIZE){
    int MIN;
    for(int j = i +1; j < SIZE; j++){
            MIN = i;
            if(arr[j] < arr[MIN]){
                MIN = j;
                swap_c(&arr[MIN], &arr[i]);
        }
    }
}
void SelectionSort_n(int arr[], int SIZE){
    for(int i = 0; i < SIZE - 1; i++){
        MIN_n(i,arr,SIZE);
    }
}
void SelectionSort_c(char arr[], int SIZE){
    for(int i = 0; i < SIZE - 1; i++){
            MIN_c(i,arr,SIZE);
    }
}
void PrintArray_n(int arr[], int SIZE){
    int i;
    for(i = 0; i < SIZE; i++)
        printf("%d  ", arr[i]);
}
void PrintArray_c(char arr[], int SIZE){
    int i;
    for(i = 0; i < SIZE; i++)
        printf("%c  ", arr[i]);
}

void PrintArray(char arr[], int SIZE){
    int i;
```

```
    for(i = 0; i < SIZE; i++)
        printf("%c  ", arr[i+1]);
}
//bubbleSortAlgorithm
void bubblesort (char *arr,int x){
    int i, j;
    for (i = 0; i <= x-1; i++){
        for (j = 0; j <= x-i-1; j++){
            if (arr[j] > arr[j+1]){
                swap_c(arr+j,arr+j+1);
            }
        }
    }

}

//binarySearchAlgorithm
void binarySearch(char arr[], int first, int last, int middle, char search)
{
    while (first <= last)
      {

          if (arr[middle] < search)
          {
              first = middle + 1;
          }

          else if (arr[middle] == search)
          {
             printf("%c is present at index %d.\n", search, middle);
             break;
          }
          else
             last = middle - 1;

          middle = (first + last)/2;
       }
       if (first > last)
        {
                printf("Not found %c is not present in the list.\n", search);
        }
}
void binarySearchi(int arr[], int first, int last, int middle, int search)
{
    while (first <= last)
      {
```

```
      if (arr[middle] < search)
      {
          first = middle + 1;
      }

      else if (arr[middle] == search)
      {
         printf("%d is present at index %d.\n", search, middle);
         break;
      }
      else
         last = middle - 1;

      middle = (first + last)/2;
    }
    if (first > last)
     {
             printf("Not found %d is not present in the list.\n", search);
     }
   return 0;
}


int main()
{
    printf("##### Enter the number of the sort that you want to use #####\n");
    printf("Press 1 for Merge Sort\nPress 2 for Selection Sort\nPress 3 for Bubble Sort\n");
    int num;
    scanf("%d", &num);
    if(num == 1){
        printf("Please insert the size of your elements then insert the elements (just independent
single integers: 0-9): \n");
        char arr[5000] = {0};
        int left, right, size;
        scanf("%d", &size);
        right = size;
        left = 0;
        printf("Hey there, Enter your elements: \n");
        for(int i = left; i <= right; i++){
            scanf("%c", &arr[i]);
        }

        /*printf("Original array: ");
        for(int j = left; j <= right; j++){
            printf("%c", arr[j+1]);
```

```
    }*/
    printf("\nSorted array: ");

    mergeSort(arr, left, right);

    for(int k = left; k <= right; k++){
        printf("%c", arr[k+1]);
    }

    int first = 0;
    int last = size;
    int middle = (first+last)/2;
    char search;
    printf("\nEnter the Number to find:\n");
    scanf("%s", &search);
    binarySearch(arr,first, last, middle, search);

    printf("\nDone :D ");

}else if(num == 2){
    int c,s;
    printf("Please insert your choice:\n\n");
    printf("1-Sort integers\t 2-Sort characters\n");
    scanf("%d",&c);
    printf("Please insert the size of your array: ");
    scanf("%d",&s);
    if(c==1){
        int arri[s];
        printf("\nInsert your elements one per line:\n");
        for(int i = 0; i < s; i++)
            scanf("%d",&arri[i]);
        SelectionSort_n(arri, s);
        printf("\n\nAfter the array of integers is sorted\n\n");
        PrintArray_n(arri, s);
        int first = 0;
        int last = s;
        int middle = (first+last)/2;
        int search;
        printf("\n\nEnter the Number to find: ");
        scanf("%d", &search);
        binarySearchi(arri,first, last, middle, search);
        }
    else{
        char arrc[s];
        printf("\nInsert your elements one per line:\n");
        for(int i = 0; i < s; i++)
```

```
                scanf("%s",&arrc[i]);
            SelectionSort_c(arrc, s);
            printf("\n\nAfter the array of characters is sorted:\n\n");
            PrintArray_c(arrc, s);
            int first = 0;
            int last = s;
            int middle = (first+last)/2;
            char search;
            printf("\n\nEnter the Character to find: ");
            scanf("%s", &search);
            binarySearch(arrc,first, last, middle, search);

            }

    }

    else if(num == 3){
        printf("Please insert the size of your elements then insert the elements: \n");
        int size;
        scanf("%d", &size);
        char arr[size];

        printf("Enter your integers/characters one per line: \n");
        for(int i = 0; i < size; i++){
            scanf("%s", arr+i);
        }

        printf("\nSorted array: ");

        bubblesort(arr, size);

        PrintArray(arr,size);


        int first = 0;
        int last = size;
        int middle = (first+last)/2;
        char search;
        printf("\nEnter the Number to find:\n");
        scanf("%s", &search);
        binarySearch(arr,first, last, middle, search);


    }
}
```

MIPS Code

```
.data
	msc: .asciiz "Enter the number of the sort that you want to use \nPress 1 for Merge Sort\nPress 2 for
	Selection Sort\nPress 3 for Bubble Sort\n"
	prompt: .asciiz  "\nEnter your characters/integers: \n"  # Prompt asking for user input
	newLine: .asciiz "\n"                                    # Newline character
	theString: .asciiz " "                                   # A fifty character string initially filled with
	whitespace
	msc2: .asciiz "\n enter the number of characters/integers \n"


	choice:        .asciiz "\nPlease select your choice: \n 1-Sort Integers\t2-Sort Characters\n"
	size:          .asciiz "\nInsert the size of the array: "
	buffer: .byte 100                    #Reserve 100 byte in the meomery for 100 charachters
	elementsI:     .asciiz "Insert the array elements,one per line  \n"
	elementsC:     .asciiz "\nInsert the array elements: "
	sorted: .asciiz "After the array is sorted:   "
	c:             .asciiz ", "


	input_size: .asciiz "\nPlease insert the size of your elements : "
	input_numbers: .asciiz "\nEnter the array elements : \n"
	arr: .word 0 #array declaration
	Sorted_Array:  .asciiz        "Sorted Array: ["
	Space:         .asciiz        ", "
	Bracket:       .asciiz        "]"
	h:      .word 0:100 #int h[100] is global



	num_search: .asciiz "\nEnter the Element to be searched: "
	found: .asciiz "\nThe Element is present in the Array at Position: "
	not_found: .asciiz "\nElement not found in the Array."
	d:             .byte '.'
	n:             .byte '\n'
.text
	main:

	    la $a0, msc     # Load address of prompt from memory into $a0
	    li $v0, 4         # Load Opcode: 4 (print string)
	    syscall           # Init syscall


	    li $v0, 5                # read the array number from the user
	    syscall
```

```
    addi $t0, $v0, 0         # number of elements=$t0
    beq $t0, 1, mergeFunction   #IF ($t0==1) call mergeFunction fun
    beq $t0, 2, Selection_Sort      #IF ($t0==2) call Selection_Sort fun
    beq $t0,3,Bubble                #IF ($t0==3) call Bubble fun
    j exit

Bubble:


    la $a0, msc2    # Load address of prompt from memory into $a0
    li $v0, 4         # Load Opcode: 4 (print string)
    syscall           # Init syscall

    li $v0, 5                # read the array number from the user
    syscall
    addi $t7, $v0, 0         # number of elements=$t0

    la $a0, prompt    # Load address of prompt from memory into $a0
    li $v0, 4         # Load Opcode: 4 (print string)
    syscall           # Init syscall



    la $a0,theString  # Load address of theString into syscall argument a0
    addi $a3,$t7,1
    move $a1,$a3        # Load sizeOfInput+1 into syscall argument a1
    li $v0,8          # Load Opcode: 8 (Read String)
    syscall

    move $s7,$t7         # s7 upper index ,Define total num of chars

    jal sort
    jal print
    j exit


sort:

    add $t0,$zero,$zero  #Initialize incrementer for outerloop
    la $s0, theString    # Load base address to theString into $t0
    add $t6,$zero,$zero  # Set index i = 0 ($t6)


    loop:   #  Outer Loop
```

```
        beq $t0,$s7,done  #Check for sentinal val and if true branch to done


        sub $t7,$s7,$t0   # Initialize upper bound of inner loop ( size - i - 1 )
        addi $t7,$t7,-1


        add $t1,$zero,$zero   #  Initialize incrementer for inner loop

        jLoop:   #  Inner Loop

            beq $t1,$t7,continue #  Check for sentinal val and if true branch to continue


            add $t6,$s0,$t1
            lb  $s1,0($t6)  #Load Array[i]
            lb  $s2,1($t6)  #Load Array[i+1]

            sgt $t2, $s1,$s2 #  If ascii(Array[i]) > ascii(Array[i+1]) then swap and store

            beq $t2, $zero, inc  #  Else,  don't swap and store
            sb  $s2,0($t6)       # swap both elements
            sb  $s1,1($t6)

            inc:

            addi $t1,$t1,1 #  increment and Jump back
            j jLoop

        continue:

        addi $t0,$t0,1   #  increment and Jump back
        j loop   # calling loop fun

print: #Prints whatever is stored inside theString

    la $a0,newLine   # Print a new line
    li $v0,4
    syscall

    add $t6,$zero,$zero # Set index i = 0 $t6

    lprint:

        beq $t6,$s7,done  #Check for sentinal val and if true
```

```
        #  Load Array[i] into t1 and print
        add $t1,$s0,$t6
        lb $a0, 0($t1)  # Load argument
        li $v0, 11      # Load opcode
        syscall         # Call syscall

        addi $t6,$t6,1  #  increment and Jump back
        j lprint        #excute the fun again

done:
    jr $ra  #continue the excution after calling the function
exit:
     li $v0,10
     syscall

Selection_Sort:
              li     $v0, 4               #Tell the system to print a string
              la     $a0, choice          #Send the string to argument $a0
              syscall                     #Call the system
              li     $v0, 5               #Tell the system to read an integer and store it in $v0
              syscall                     #Call the system
              move   $s1, $v0             #Store the input value(size) of $v0 into $s1
              move   $s2, $zero           #Initialize $s2 by zero
              move   $s3, $zero           #Initialize $s3 by zero
              addi   $s2, $s2, 1          #Increment $s2 by 1
              addi   $s3, $s2, 1          #Increment $s2 by 1 and store the value into $s3
              beq    $s1, $s2, Selection_Sort_Int #Check if $s1(choice) = $s2 then go to
Selection_Sort_Int
              beq    $s1, $s3, Selection_Sort_Char#Check if $s1(choice) = $s3 then go to
Selection_Sort_Char

Selection_Sort_Int:
              li     $v0, 4               #Tell the system to print a string
              la     $a0, size            #Send the string to argument $a0
              syscall                     #Call the system
              li     $v0, 5               #Tell the system to read an integer and store it in $v0
              syscall                     #Call the system
              move   $s2, $v0             #Store the input value(size) of $v0 into $s2
              li     $v0, 4               #Tell the system to print a string
              la     $a0, elementsI       #Send the string to argument $a0
              syscall                     #Call the system
              move   $s1, $zero           #Initialize $s1 by zero
run_loopI:
              bge    $s1, $s2, exit_loopI  #Check if $s1(counter) >= $s2(size) then go to exit_loop
              sll    $t0, $s1, 2          #Shift left $s1(counter)  by 2 and store the value in
$t0(index of ith element); $s1*4(bytes of integer)
```

```
                add     $t1, $t0, $sp  #Add $t0(index of ith element) to $sp(stack pointer) and store the
value in $t1(address of ith element)
                li      $v0, 5                #Tell the system to read an integer and store it in $v0
                syscall                       #Call the system
                sw      $v0, ($t1)            #Save the input value of $v0 in index i
                li      $v0, 4                #Tell the system to print a string
                la      $a0, n                #Send the string to argument $a0
                syscall                       #Call the system
                addi    $s1, $s1, 1           #Increment the counter by 1
                j       run_loopI             #Jump to run_loop again

exit_loopI:
                move    $a0, $sp              #Copy the address of $sp(stack pointer) into argument $a0
                move    $a1, $s2              #Copy the value of $s2(size) into argument $a1
                jal     Selection_SortI       #Call the sort function
                li      $v0, 4                #Tell the system to print a string
                la      $a0, sorted           #Send the string to argument $a0
                syscall                       #Call the system

printI:
                move    $s1, $zero            #Initialize $s1 by zero
ploopI:
                bge     $s1, $s2, exit_printI#Check if $s1(counter) >= $s2(size) then go to exit_print
                sll     $t0, $s1, 2           #Shift left $s1(counter)  by 2 and store the value in
$t0(index of ith element); $s1*4(bytes of integer)
                add     $t1, $t0, $sp  #Add $t0(index of ith element) to $sp(stack pointer) and store the
value in $t1(address of ith element)
                lw      $a0, ($t1)            #Store the returned value in argument $a0
                li      $v0, 1                #Tell the system to print an integer
                syscall                       #Call the system
                addi    $s1, $s1, 1           #Increment the counter by 1
                bge     $s1, $s2, exit_tempI  #Check if $s1(counter) >= $s2(size) then go to exit_temp
                li      $v0, 4                #Tell the system to print a string
                la      $a0, c                #Send the string to argument $a0
                syscall                       #Call the system
exit_tempI:
                j       ploopI                #Jump to ploop again

exit_printI:
                li      $v0, 4                #Tell the system to print a string
                la      $a0, d                #Send the string to argument $a0
                syscall                       #Call the system
                li      $v0, 10       #Exit the program
                syscall                       #Call the system

Selection_SortI:
```

```
                addi    $sp, $sp, -16  #Push 5 elements in the stack
                sw      $s0, 0($sp)           #Save value of $s0 in index 0
                sw      $s1, 4($sp)           #Save value of $s1 in index 1
                sw      $s2, 8($sp)           #Save value of $s2 in index 2
                sw      $ra, 12($sp)          #Save the return address in index 3
                move    $s0, $a0              #Copy the base address of the array to $s0
                move    $s1, $zero            #Initialize $s1 by zero
                subi    $s2, $a1, 1           #Subtract 1 from $a1 and store the value in $s2

    Sort_loopI:
                bge     $s1, $s2, Selection_Sort_exitI      #Check if $s1(counter) >= $s2(size-1) then go
    to Selection_Sort_exit
                move    $a0, $s0              #Copy the base address of the array to $a0
                move    $a1, $s1              #Copy the value of $s1(counter) into argument $a1
                move    $a2, $s2              #Copy the value of $s2(size-1) into argument $a2
                jal     minI                  #Call min function
                move    $a2, $v0              #Store the returned value(index of min) of $v0 into  argument
    $a2
                jal     swapI                 #Call the swap function
                addi    $s1, $s1, 1           #Increment the counter by 1
                j       Sort_loopI            #Jump to Sort_loop again

    Selection_Sort_exitI:
                lw      $s0, 0($sp)           #Load value at index 0 into $s0
                lw      $s1, 4($sp)           #Load value at index 1 into $s1
                lw      $s2, 8($sp)           #Load value at index 2 into $s2
                lw      $ra, 12($sp)          #Load the return address at index 3 into $ra
                addi    $sp, $sp, 16          #Pop 4 elements from the stack (i.e. adjust the stack pointer)
                jr      $ra                   #Return address (i.e. exit the funcion)

    minI:
                move    $t0, $a0              #Copy the base address of the array to $t0
                move    $t1, $a1              #Copy the value of argument $a1(counter) into $t1
                move    $t2, $a2              #Copy the value of argument $a2(size-1) into $t2
                move    $t5, $t1              #Copy the value of $t1(counter) into $t5
                sll     $t3, $t1, 2           #Shift left $t1(counter)  by 2 and store the value in
    $t3(index of ith element); $t1*4(bytes of integer)
                add     $t3, $t3, $t0  #Add $t3(index of ith element) to $t0(base address) and store the
    value in $t3(address of ith element)
                lw      $t4, ($t3)            #Store the returned value in $t4(current element/min)

    min_loopI:
                bgt     $t5, $t2, min_exitI   #Check if $t5(counter) >= $t2(size-1) then go to min_exit
                sll     $t6, $t5, 2           #Shift left $t5(counter)  by 2 and store the value in
    $t6(index of ith element); $t5*4(bytes of integer)
                add     $t6, $t6, $t0  #Add $t6(index of ith element) to $t0(base address) and store the
```

```
value in $t6(address of ith element)
                lw      $t7, ($t6)            #Store the returned value in $t7(ith element)
                bge     $t7, $t4, chk_exitI   #Check if $t7(ith element) >= $t2(current element/min) then go
to min_exit
                move    $t1, $t5              #Copy the value(address of new min) of  $t5 into $t1
                move    $t4, $t7              #Copy the value of $t7(new min) into $t4(old min)

chk_exitI:
                addi    $t5, $t5, 1           #Increment the counter by 1
                j       min_loopI             #Jump to min_loop again

min_exitI:
                move    $v0, $t1              #Return the index of the min character in $v0
                jr      $ra                   #Return address (i.e. exit the funcion)

swapI:
                sll     $t1, $a1, 2           #Shift left $a1(counter)  by 2 and store the value in $t1;
$a1*4(bytes of integer)
                add     $t1, $t1, $a0         #Add $t1(index of ith element) to $a0(base address) and store
the value in $t1(address of ith element)
                sll     $t2, $a2, 2           #Shift left $a2(index of min)  by 2 and store the value in
$t2; $a2*4(bytes of integer)
                add     $t2, $t2, $a0  #Add $t1(index of min) to $a0(base address) and store the value in
$t2(address of of min)
                lw      $t0, ($t1)            #Load the integer of index $t1(counter) in $t0; $t0 =
a[counter]
                lw      $t3, ($t2)            #Load character of index $t2(minimum) in $t3; $t3 = a[min]
                sw      $t3, ($t1)            #Store integer(minimum) of $t3 at index $t1(counter);
s[counter] = s[min]
                sw      $t0, ($t2)            #Store integer(counter) of $t0 at index $t2(minimum); s[min] =
s[counter]
                jr      $ra                   #Return address       (i.e. exit the funcion)

Selection_Sort_Char:
                li      $v0, 4                #Tell the system to print a string
                la      $a0, elementsC        #Send the string to argument $a0
                syscall                       #Call the system
                li      $v0, 8                #Tell the system to read a string
                la      $a0, buffer           # Read the string
                li      $a1, 100              #Allocate  space in the meomery for the string
                syscall                       #Call the system
                la      $t0, buffer           #Load address of the buffer to $t0
run_loopC:
                lb      $t1, ($t0)            #Load from $t0 the ith character of the string in $t1; $t1 =
s[i]
                beq     $t1, '\0', exit_loopC #Check if it is the end of string or not
```

```
                addi    $t0, $t0, 1           #Increment the index of the current position by 1 (i++)
                addi    $s0, $s0, 1           #Increment the counter (count number of characters in the
string) by 1
                j       run_loopC             #Jump to run_loop again
exit_loopC:
                subi    $s0, $s0, 1           #Decrease the value of $s0 by 1
                move    $a0, $s0              #Copy the value of $s0(num of characters in string) into
argument $a0
                la      $a1, buffer           #Load address of the buffer(unsorted string) to argument $a1
                jal     Selection_SortC       #Call the sort function
                li      $v0, 4                #Tell the system to print a string
                la      $a0, sorted           #Send the string to argument $a0
                syscall                       #Call the system
                la      $a1, buffer           #Load address of the buffer(sorted string) to argument $a1
                move    $a2, $s0              #Copy the value of $s0(counter) into argument $a2
                jal     printC                #Call print function
                li      $v0, 10        #Exit the program
                syscall                       #Call the system
printC:
                addi    $sp, $sp, -8          #Push 2 elements in the stack
                sw      $s1, 0($sp)           #Save value of $s1 in index 0
                sw      $s2, 4($sp)           #Save value of $s2 in index 1
                move    $s1, $zero            #Initialize $s1 by zero
                move    $s2, $a2              #Copy the value of argument $a2(counter) into $s2
ploopC:
                bge     $s1, $s2, exit_printC#Check if $s1(counter) >= $s2(num of characters in string)
then go to exit_print
                lb      $t1, ($a1)            #Load ith character of the string in $t1; $t1 = s[i]
                li      $v0, 11        #Tell the system to print a character
                la      $a0, ($t1)            #Load the address of $t1 into argument $a0
                syscall                       #Call the system
                addi    $a1, $a1, 1           #Increment the index of the current position by 1 (i++)
                addi    $s1, $s1, 1           #Increment the counter by 1
                bge     $s1, $s2, exit_tempC  #Check if $s1(counter) >= $s2(num of characters in string)
then go to exit_temp
                li      $v0, 4                #Tell the system to print a string
                la      $a0, c                #Send the string to argument $a0
                syscall                       #Call the system
                j       ploopC                #Jump to ploop again
exit_tempC:
                j       ploopC                #Jump to ploop again
exit_printC:
                lw      $s1, 0($sp)           #Load value at index 0 into $s1
                lw      $s2, 4($sp)           #Load value at index 1 into $s2
                addi    $sp, $sp, 8           #Pop 2 elements from the stack (i.e. adjust the stack pointer)
                li      $v0, 4                #Tell the system to print a string
```

```
                la      $a0, d                #Send the string to argument $a0
                syscall                       #Call the system
                jr      $ra                   #Return address (i.e. exit the funcion)

Selection_SortC:
                addi    $sp, $sp, -16  #Push 4 elements in the stack
                sw      $s1, 0($sp)           #Save value of $s1 in index 0
                sw      $s2, 4($sp)           #Save value of $s2 in index 1
                sw      $s3, 8($sp)           #Save value of $s3 in index 2
                sw      $ra, 12($sp)          #Save the return address in index 3
                move    $s3, $zero            #Initialize $s3 by zero
                move    $s2, $a0              #Copy the value of argument $a0(num of characters in string)
into $s2
Sort_loopC:
                bge     $s3, $s2, Selection_Sort_exitC      #Check if $s3(counter) >= $s2(num of characters
in string) then go to Selection_Sort_exit
                la      $a1, buffer           #Load address of the buffer(unsorted string) to argument $a1
                move    $a0, $s3              #Copy the value of $s3(counter) into  argument $a0
                add     $a1, $a1, $s3         #Add $s3(counter) to argument $a1 and store the value in $a1
                move    $a2, $a1              #Copy the value(address) of argument $a1 into  argument $a2
                move    $a3, $s2              #Copy the value of $s2(num of characters in string) into
argument $a3
                jal     minC                  #Call min function
                move    $a2, $v0              #Store the returned value(index of min) of $v0 into  argument
$a2
                jal     swapC                 #Call the swap function
                addi    $s3, $s3, 1           #Increment the counter by 1
                j       Sort_loopC            #Jump to Sort_loop again
Selection_Sort_exitC:
                lw      $s1, 0($sp)           #Load value at index 0 into $s1
                lw      $s2, 4($sp)           #Load value at index 1 into $s2
                lw      $s3, 8($sp)           #Load value at index 2 into $s3
                lw      $ra, 12($sp)          #Load the return address at index 3 into $ra
                addi    $sp, $sp, 16          #Pop 4 elements from the stack (i.e. adjust the stack pointer)
                jr      $ra                   #Return address (i.e. exit the funcion)
minC:
                move    $s1, $a0              #Copy the value of argument $a0(counter) into $s1
                move    $t1, $a2              #Copy the value(address) of argument $a2 into $t1
                move    $t3, $a3              #Copy the value of argument $a3(num of characters in string)
into $t3
                move    $t4, $t1              #Copy the value(address) of $t1 into $t4
                lb      $t5, ($t1)            #Load from $t1 the ith character of the string in $t5(current
character/min); $t5(min) = s[i]
min_loopC:
                bge     $s1, $t3, min_exitC   #Check if $s1(counter) >= $t3(num of characters in string)
then go to min_exit
```

```
                lb      $t6, ($t4)            #Load from $t4 the ith character of the string in $t6; $t6 =
s[i]
                bge     $t6, $t5, chk_exitC   #Check if $t6(ith character) >= $t5(current character/min)
then go to chk_exit
                move    $t1, $t4              #Copy the value(address of new min) of  $t4 into $t1
                move    $t5, $t6              #Copy the value of $t6(new min) into $t5(old min)
                addi    $s1, $s1, 1           #Increment the counter by 1
                addi    $t4, $t4, 1           #Increment the index of the current position by 1 (i++)
                j       min_loopC             #Jump to min_loop again
chk_exitC:
                addi    $s1, $s1, 1           #Increment the counter by 1
                addi    $t4, $t4, 1           #Increment the index of the current position by 1 (i++)
                j       min_loopC             #Jump to min_loop again
min_exitC:
                move    $v0, $t1              #Return the index of the min character in $v0
                jr      $ra                   #Return address (i.e. exit the funcion)
swapC:
                lb      $t0, ($a1)            #Load the character of index $a1(counter) from the string in
$t0; $t0 = s[counter]
                lb      $t3, ($a2)            #Load character of index $a2(minimum) from the string in $t3;
$t3 = s[min]
                sb      $t3, ($a1)            #Store character(minimum) of $t3 at index $a1(counter);
s[counter] = s[min]
                sb      $t0, ($a2)            #Store character(counter) of $t0 at index $a2(minimum); s[min]
= s[counter]
                jr      $ra                   #Return address       (i.e. exit the funcion)



mergeFunction:

        li $v0, 4
        la $a0, input_size           # print the array message
        syscall

        li $v0, 5                    # read the array number from the user
        syscall

        addi $t0, $v0, 0             # number of elements=$t0
        addi $t1, $zero, 0              # iterator

        li $v0, 4
        la $a0, input_numbers        # print the array numbers
        syscall

        la $s1, arr                  # array adress= $s1
```

```
input_loop:
        beq $t1, $t0, exit_input     # if t1(i) = size if array exit from the while loop
        li $v0,5                     # read all numbers of array from user
        syscall
        sw $v0, 0($s1)               # store the value in the s1 variable
        addi $s1, $s1, 4             # s1 = s1 + 4
        addi $t1, $t1, 1             # i = i + 1
        j input_loop                 # recursion loop (function call)

exit_input:

        la $a0, arr           # load address of arr to $a0 as an argument
        addi $a1, $zero, 0    # $a1 = low
        addi $a2, $t0, -1     # $a2 = high

        jal Mergesort         # Go to MergeSort
        #la  $a0, Sorted_Array       # Print prompt: "Sorted Array: ["
        #li  $v0, 4           # MIPS call for printing prompts
        #syscall
        jal Print             # Go to Print to print the sorted array
        #la  $a0, Bracket     # Prints the closing bracket for the array
        #li  $v0, 4           # MIPS call for printing prompts
        #syscall
        jal Exit_input
        li  $v0, 10           # Done!
        syscall

Mergesort:
        slt $t0, $a1, $a2     # if low < high then $t0 = 1 else $t0 = 0
        beq $t0, $zero, Return       # if $t0 = 0, go to Return

        #lookHere :D
        addi, $sp, $sp, -16   # Make space on stack for 4 items
        sw, $ra, 12($sp)      # save return address
        sw, $a1, 8($sp)              # save value of low in $a1
        sw, $a2, 4($sp)              # save value of high in $a2
        #
        add $s0, $a1, $a2     # mid = low + high
        #when shifting right happen one digit equals to division by 2
        sra $s0, $s0, 1              # mid = (low + high) / 2
        sw $s0, 0($sp)               # save value of mid in $s0
        add $a2, $s0, $zero   # make high = mid to sort the first half of array
        jal Mergesort         # recursive call to MergeSort

        lw $s0, 0($sp)        # load value of mid that's saved in stack
```

```
        addi $s1, $s0, 1      # store value of mid + 1 in $s1
        add $a1, $s1, $zero   # make low = mid + 1 to sort the second half of array
        lw $a2, 4($sp)               # load value of high that's saved in stack
        jal Mergesort         # recursive call to MergeSort

        #Note that $a0 have the array address which is global

        lw, $a1, 8($sp)       # load value of low that's saved in stack
        lw, $a2, 4($sp)       # load value of high that's saved in stack
        lw, $a3, 0($sp)       # load value of mid that's saved in stack and pass it to $a3 as an argument
for Merge
        jal Merge             # Go to Merge

        lw $ra, 12($sp)              # restore $ra from the stack
        addi $sp, $sp, 16     # restore stack pointer
        jr  $ra

Return:
        jr $ra                # return to calling routine

Merge:
        add  $s0, $a1, $zero  # $s0 = i; i = low
        add  $s1, $a1, $zero  # $s1 = k; k = low
        addi $s2, $a3, 1      # $s2 = j; j = mid + 1

While1:
        blt  $a3,  $s0, While2       # if mid < i then go to next While loop
        blt  $a2,  $s2, While2       # if high < j then go to next While loop
        j  If                 # if i <= mid && j <=high

If:     #Assign array values to registers
        sll  $t0, $s0, 2      # $t0 = i*4
        add  $t0, $t0, $a0    # add offset to the address of a[0]; now $t2 = address of a[i]
        lw   $t1, 0($t0)      # load the value at a[i] into $t1
        sll  $t2, $s2, 2      # $t1 = j*4
        add  $t2, $t2, $a0    # add offset to the address of a[0]; now $t2 = address of a[j]
        lw   $t3, 0($t2)      # load the value of a[j] into $t3
        #Start of if-else statment
        blt  $t3, $t1, Else   # if a[j] < a[i], go to Else
        la   $t4, c           # Get start address of c
        sll  $t5, $s1, 2      # k*4
        add  $t4, $t4, $t5    # $t4 = c[k]; $t4 is address of c[k]
        sw   $t1, 0($t4)      # c[k] = a[i]
        addi $s1, $s1, 1      # k++
        addi $s0, $s0, 1      # i++
        j    While1           # Go to next iteration
```

```
Else:
        sll  $t2, $s2, 2      # $t1 = j*4
        add  $t2, $t2, $a0    # add offset to the address of a[0]; now $t2 = address of a[j]
        lw   $t3, 0($t2)      # $t3 = whatever is in a[j]
        la   $t4, c           # Get start address of c
        sll  $t5, $s1, 2      # k*4
        add  $t4, $t4, $t5    # $t4 = c[k]; $t4 is address of c[k]
        sw   $t3, 0($t4)      # c[k] = a[j]
        addi $s1, $s1, 1      # k++
        addi $s2, $s2, 1      # j++
        j    While1           # Go to next iteration

While2:
        blt  $a3, $s0, While3        # if mid < i
        sll $t0, $s0, 2              # # $t6 = i*4
        add $t0, $a0, $t0     # add offset to the address of a[0]; now $t6 = address of a[i]
        lw $t1, 0($t0)        # load value of a[i] into $t7
        la  $t2, c            # Get start address of c
        sll $t3, $s1, 2         # k*4
        add $t3, $t3, $t2     # $t5 = c[k]; $t4 is address of c[k]
        sw $t1, 0($t3)               # saving $t7 (value of a[i]) into address of $t5, which is c[k]
        addi $s1, $s1, 1      # k++
        addi $s0, $s0, 1      # i++
        j While2              # Go to next iteration


While3:
        blt  $a2,  $s1, For_Initializer     #if high < j then go to For loop
        sll $t2, $s2, 2       # $t6 = j*4
        add $t2, $t2, $a0     # add offset to the address of a[0]; now $t6 = address of a[j]
        lw $t3, 0($t2)        # $t7 = value in a[j]

        la  $t4, c            # Get start address of c
        sll $t5, $s1, 2              # k*4
        add $t4, $t4, $t5     # $t5 = c[k]; $t4 is address of c[k]
        sw $t3, 0($t4)        # $t4 = c[k]; $t4 is address of c[k]
        addi $s1, $s1, 1      # k++
        addi $s2, $s2, 1      # j++
        j While3              # Go to next iteration

For_Initializer:
        add  $t0, $a1, $zero  # initialize $t0 to low for For loop
        addi $t1, $a2, 1      # initialize $t1 to high+1 for For loop
        la   $t4, c           # load the address of array c into $t4
        j    For
```

```
For:
        slt $t7, $t0, $t1     # $t7 = 1 if $t0 < $t1
        beq $t7, $zero, sortEnd      # if $t7 = 0, go to sortEnd
        sll $t2, $t0, 2       # $t0 * 4 to get the offset
        add $t3, $t2, $a0     # add the offset to the address of a => a[$t3]
        add $t5, $t2, $t4     # add the offset to the address of c => c[$t5]
        lw  $t6, 0($t5)              # loads value of c[i] into $t6
        sw $t6, 0($t3)        # save the value at c[$t0] to a[$t0]; a[i] = c[i]
        addi $t0, $t0, 1      # increment $t0 by 1 for the i++ part of For loop
        j For                 # Go to next iteration

sortEnd:
        jr $ra                # return to calling routine
Print:
        add $t0, $a1, $zero   # initialize $t0 to low
        add $t1, $a2, $zero   # initialize $t1 to high
        la  $t4, arr          # load the address of the array into $t4
Print_Loop:
        blt  $t1, $t0, Exit   # if $t1 < $t0, go to exit
        sll  $t3, $t0, 2      # $t0 * 4 to get the offset
        add  $t3, $t3, $t4    # add the offset to the address of array to get array[$t3]
        lw   $t2, 0($t3)      # load the value at array[$t3] to $t2
        move $a0, $t2         # move the value to $a0 for printing
        li   $v0, 1           # the MIPS call for printing the numbers
        syscall

        addi $t0, $t0, 1      # increment $t0 by 1 for the loop
        #la   $a0, Space              # prints a comma and space between the numbers
        #li   $v0, 4          # MIPS call to print a prompt
        #syscall
        j    Print_Loop              # Go to next iteration of the loop
Exit_input:
        #input for searching elements.. $a3=number to be searched
        li $v0, 4
        la $a0, num_search           # print (enter the value to be search)
        syscall

        li $v0, 5                    # read the value (search) from the user
        syscall

        # Our inputs
        addi $a3, $v0, 0             # a3 = number to be searched
        la $a0, arr                  # a0 = array
        addi $a1, $zero, 0           # a1 = First
        addi $a2, $t0, -1            # a2 = Last
```

```
        jal binary_search            #function call
        #back to the main from funtion
        addi $t7, $zero, -1           # t7 = -1
        beq $v1, $t7, nott           # if v1 == t7 call the function nott
        li $v0, 4
        la $a0, found                # print that the element that be searched is in the array
        syscall
        li $v0, 1
        addi $a0, $v1, 0             # print the position of the element
        syscall
        li $v0, 10
        syscall

nott:
        li $v0, 4
        la $a0, not_found
        syscall
        li $v0, 10
        syscall

binary_search:

        la $a0, arr                  # a0 = array
        addi $s2, $zero, 0           # s2 = 0 = middle
        slt $t1, $a2, $a1            # t1 = 1 if ( a2 < a1 )    while (first < last)
        beq $t1, $zero, if1          # if ( t1 == 0 ) call function (if1)
        addi $v1, $zero, -1          # v1 += -1
        jr $ra

if1:
        add $s2, $a2, $a1            # s2 = a2 + a1 ( middle = last + first )

        addi $t9, $zero, 2           # t9=2 so that i could divide mid with by 2
        div $s2, $t9                 # s2 / 2  ( div the middle on 2 )
        mflo $s2                     # move from LOW to s2
        mflo $s3                     # move from LOW to s3
        mul $s3, $s3, 4              # s3 = 4 * s3
        add $a0, $a0, $s3            # array (a0) = a0 +s3
        lw $s4, 0($a0)               # s4 = the first elemnt in array
        bne $s4, $a3, if2            # if array(i) in not equal the searched number then call if2
        addi $v1, $s2, 1             # vi = s2 + 1
        jr $ra

if2:
        slt $t5, $a3, $s4            # t5 = 1 if ( a3 < s4 )
        beq $t5, $zero, else         # if ( t5 = 0 ) call function else
```

```
        addi $a2, $s2, -1            # a2 = s2 -1
        addi $sp, $sp, -4            # sp = sp - 4
        sw $ra, 0($sp)
        jal binary_search

        j exit1                      # jump to function exit

else:
        addi $a1, $s2, 1             # a1 = s2 +1
        addi $sp, $sp, -4            # sp = sp - 4
        sw $ra, 0($sp)
        jal binary_search
        j exit1                      # jump to function exit

exit1:
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra


Exit:
        jr $ra                # jump to the address in $ra; Go back to main
```