***PROJECT REPORT***

* ***CLASS : BSE 3A (SOFTWARE)***
* ***SUMISSION DATE:22/JAN2021***
* ***SUBMITTED TO: SIR REHAN BAIG***



SESSI-ON

2

0

2

0



***PROJECT TITTLE:***

***BUBBLE SORTING AND BINARY SEARCH***

* GROUP MEMBER:03

|  |  |  |  |
| --- | --- | --- | --- |
| ***S.NO*** | ***MEMBER NAMES*** | ***ENROLLMENT*** | ***STRENGHT*** |
| ***01*** | ***SHAHWAIZ HASSAN*** | ***02-131192-006*** | ***We’ll be working***  ***collectively as a***  ***team in the entire***  ***Project.*** |
| ***02*** | ***ABDUL REHMAN*** | ***02-131192-064*** |
| ***03*** | ***SANNIA AMIN*** | ***02-131192-068*** |

* EXPECTED SUBMISSION DATE:22/01/2021



***PROJECT DESCRIPTION:***

|  |
| --- |
| * ***BINARY SEARCH AND BUBBLE SORT*** |
| Bubble sort is a sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted. Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.  **Example**: First Pass: ( 5 1 4 2 8 ) –> ( 1 5 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1  A binary search is an advanced type of search algorithm that finds and fetches data from a sorted list of items. Its core working principle involves dividing the data in the list to half until the required value is located and displayed to the user in the search result. Binary search is commonly known as a **half-interval search** or a **logarithmic search**.  Here's a step-by-step description of using binary search to play the guessing game:  Let min = 1*min*=1m, i, n, equals, 1 and max = n*max*=*n*m, a, x, equals, n.   1. Guess the average of max*max*m, a, x and min*min*m, i, n, rounded down so that it is an integer. 2. If you guessed the number, stop. You found it! 3. If the guess was too low, set min*min*m, i, n to be one larger than the guess. 4. If the guess was too high, set max*max*m, a, x to be one smaller than the guess. 5. Go back to step two. |

***CODE DESCRIPTION:***

* **JAL:** we use jal unconditional branch because we want to link and jump without any condition and we link the no of array with element of array and also during sorting and binary search we link all these statement by using jal branch.
* **Jr $ra :** WE use this statement for storing the address of next instruction.
* We set two temporary register $t0 and $t1 in which we increment the index of array and in $t1 which act a counter.
* **Lw:** we use this statement for load a word from memory to a register according to the index and element value that user will give
* **Sw**: we use this because it store a register into a memory through which we link offset, register(holding address) and source of information.
* **Ble:** we use this branch for set a if then else condition.
* **Bgt:** this branch is useful for checking the inputs because it check the value is it greater than?
* **Addi && Sub:** by using this we make a loop and check the values when user give search values

***PROJECT CODE***

############################### DATA SEGMENT PORTION IS GIVEN BELOW ################

.data

ln: .asciiz"=======================================\n"

pre: .asciiz "PREPARED BY:SANNIA & SHAWAIZ $ ABDUL REHMAN\n"

lin: .asciiz"=======================================\n"

proj: .asciiz "\*HELLOW WELCOME TO OUT SORTED SEARCH PROJECT\n"

lini: .asciiz"=======================================\n"

amount\_num: .asciiz "HOW MANY NUMBER DO YOU WANT TO PUT IN ARRAY?\*"

num: .asciiz "\n\*ENTER THE INTEGER VALUE\*: "

nl: .asciiz "\n"

array\_contains: .asciiz "\nARRAY CONATIN THE FOLLOWING NUMBER:> \n"

complete: .asciiz "\n THANKYOU PROGRAM IS COMPLETED!"

.align 2

array: .space 40

search\_num: .asciiz "\n\*\* ENTER THE NUMBER WHICH YOU WANT TO SEARCH\*\*:> "

f: .asciiz " \n---------CONGRATULATION! YOUR ENTERD WORD IS FOUND IN GIVEN ARRAY :)-----AT THE INDEX NO IS:>--"

cl\_bracket: .asciiz " )"

nf: .asciiz " \n---------SORRY! YOUR ENTERD WORD IS NOT FOUND IN GIVEN ARRAY :)-------"

############################# CODE SEGMENT PORTION IS GIVEN BELOW #################################################

.text

.globl main

la $a0,ln

li $v0,4

syscall

la $a0,pre

li $v0,40

syscall

la $a0,lin

li $v0,4

syscall

la $a0,proj

li $v0,4

syscall

la $a0,lini

li $v0,4

syscall

**MAIN SECTION CODE**

main:

jal enter\_amount # Prompts the user to enter the amount of integers to be entered.

jal enter\_num # Enter the integers into the array.

jal contains # Prints the contents of the array.

jal sort # Sorts the integers in the array using a bubble sort.

jal contains # Prints the sorted contents of the array.

jal search\_for # Enter the integer you're searching for.

jal search # Searches the array using a iterative binary search.

jal search\_results # Prints the results of the search for the integer.

j done # Exit

**AMOUNT SECTION CODE**

enter\_amount:

la $a0, amount\_num

li $v0, 4

syscall

li $v0, 5

syscall

move $t0, $v0

li $t1, 0

li $t2, 1

jr $ra

**NUMBER SECTION CODE**

enter\_num:

addi $t2, $t2, 1

la $a0, num

li $v0, 4

syscall

li $v0, 5

syscall

sw $v0, array($t1)

addi $t1, $t1, 4

ble $t2, $t0, enter\_num

jr $ra

**VALUE CONATINE SECTION CODE**

contains:

li $t1, 0

li $t2, 1

la $a0, array\_contains

li $v0, 4

syscall

print\_array:

addi $t2, $t2, 1

lw $a0, array($t1)

li $v0, 1

syscall

la $a0, nl

li $v0, 4

syscall

addi $t1, $t1, 4

ble $t2, $t0, print\_array

jr $ra

**BUBBLE SORT CODE**

sort:

li $t2, 0

outer:

addi $t2, $t2, 1

la $a1, array

li $t1, 0

sub $t3, $t0, 1

addi $t4, $t2, 1

ble $t2, $t3, inner

jr $ra

inner:

lw $t5, 0($a1)

lw $t6, 4($a1)

bgt $t5, $t6, swap

j continue

swap:

sw $t6, 0($a1)

sw $t5, 4($a1)

continue:

addi $a1, $a1, 4

addi $t4, $t4, 1

bgt $t4, $t0, outer

j inner

**BINARY SEARCH CODE**

search\_for:

la $a0, search\_num

li $v0, 4

syscall

li $v0, 5

syscall

move $t0, $v0

li $t2, 0

li $t4, 2

li $t6, 4

jr $ra

search:

bgt $t2, $t3, return

middle\_num:

add $t7, $t2, $t3

div $t7, $t4

mflo $t8

mult $t8, $t6

mflo $t1

lw $t5, array($t1)

bgt $t5, $t0, lower

blt $t5, $t0, upper

li $s0, 1

jr $ra

upper:

add $t2, $t8, 1

j search

lower:

sub $t3, $t8, 1

j search

return:

li $s0, 0

jr $ra

search\_results:

beqz $s0, not\_found

found:

move $a0, $t0

li $v0, 1

syscall

la $a0, f

li $v0, 4

syscall

div $t1, $t6

mflo $a0

li $v0, 1

syscall

la $a0, cl\_bracket

li $v0, 4

syscall

jr $ra

not\_found:

move $a0, $t0

li $v0, 1

syscall

la $a0, nf

li $v0, 4

syscall

jr $ra

done:

la $a0, complete

li $v0, 4

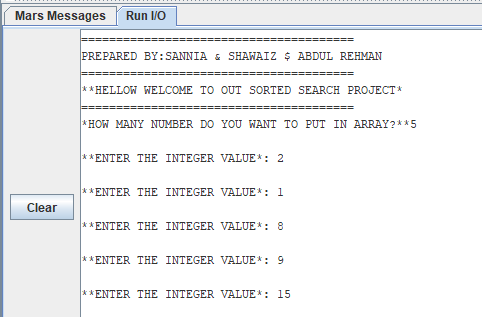
syscall

li $v0, 10

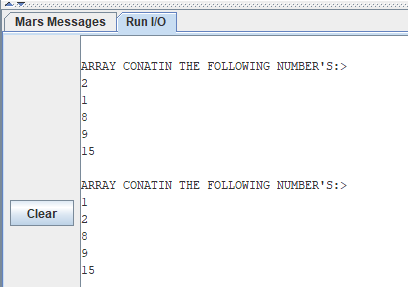
syscall

**

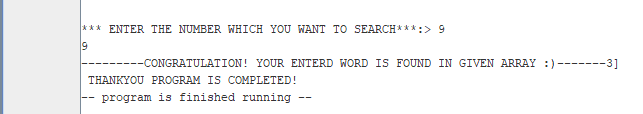
**GIVING INPUT**



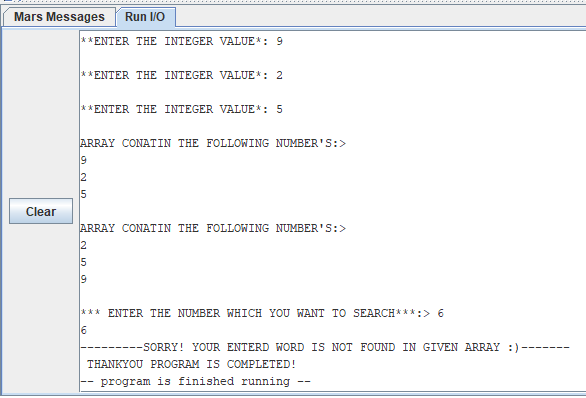
**AFTER SORTING**



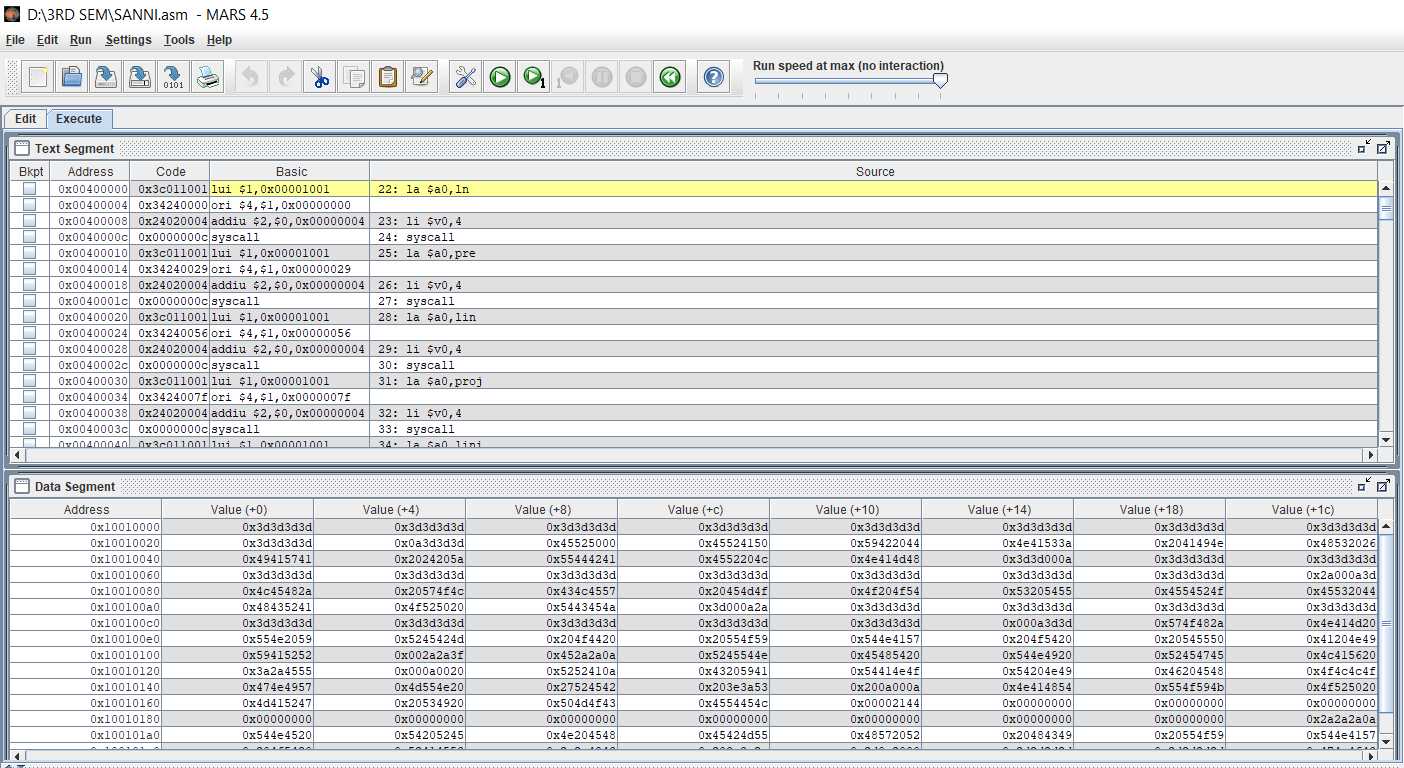
**AFTER SEARCHING**



**WHEN VALUE NOT FOUND**



**AFTER ASSEMBLE**



**REGISTER AREA**

