```
        # Program to utilize Binary Search with Quicksort
        # Written by Gavin Osborn
        # Final MIPS Project | CSCI-210 | 11/28/2022

        .data
        .align  2
        values: .word 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 37, 17, 16, 13, 18, 39, 42, 54, 55, 33, 25, 26, 27, 28,
29, 210, 211, 212, 213, 214, 237, 217, 216, 213, 218, 239, 242, 254, 255, 233, 105, 106, 107, 108, 109, 110, 111,
112, 113, 114,
        valueCount: .word 50
        unsortedPrompt: .asciiz "\n The unsorted list: "
        sortedPrompt: .asciiz "\n The sorted list: "
        binarySearchInputPrompt: .asciiz "\n Enter the value you want to search: "
        binarySearchOutputPrompt: .asciiz "\n Index of Value (-1 if not found): "
        comma: .asciiz ","
#--------------- Usual stuff at beginning of main -----------------
        .text
       .globl main
main:
#------------------------- function body -------------------------

#------ Sort List

        #------ Display unsorted list
        la  $t0, unsortedPrompt # t0: address to string
    move    $a0, $t0        # load t0 into syscall parameter
    li  $v0, 4          # call print string
    syscall
        la  $a0, values
        lw  $a1, valueCount
        jal printList

        #------ sort list
        la  $a0, values # a0: array pointer
        lw  $a1, valueCount # a1: size of array

        move    $a2, $0 # a2: start of array
        move    $a3, $a1
        addi    $a3, $a3, -1 # a3: valuesCount-1
        jal     quickSort # driver call to quickSort

        #------ Display results
        la  $t0, sortedPrompt   # t0: address to string
    move    $a0, $t0        # load t0 into syscall parameter
    li  $v0, 4          # call print string
    syscall
        la  $a0, values
        lw  $a1, valueCount
        jal printList

#------ Binary Search Function Call
binarySearchInput:
    # ask for new search, or offer to cancel
    la  $t0, binarySearchInputPrompt    # t0: address to string
    move    $a0, $t0        # load t0 into syscall parameter
    li  $v0, 51         # call print int w/ message
    syscall
    # check inputs
    # a0: int read
    # a1: status value - incorrect parse, cancel, no input
    beq $a1, -1, binarySearchInput
    beq $a1, -2, endProgram
    beq $a1, -3, binarySearchInput

startBinarySearch:
        #------- Binary Search
        move    $a1, $a0 # a1 - search value
```

```
        la  $a0, values # a0: array pointer
        move    $a2, $0 # a2 - start index of given section for searching
        lw  $a3, valueCount # a3: size of array
        addi    $a3, $a3, -1 # a3: valuesCount-1
        jal binarySearch # BinarySearch(arr, x, start, mid-1)

        #------- Binary Search Results

        # print result
        la  $t0, binarySearchOutputPrompt   # t0: address to string
    move    $a0, $t0        # load t0 into syscall parameter
    move    $a1, $v0
    li  $v0, 56         # call print int w/ message
    syscall
    j binarySearchInput

endProgram:
#----------------- Usual stuff at end of main ---------------------
        li      $v0, 10
        syscall
#*****************************************************************



#*****************************************************************
        # binarySearch function
        # Conducts a binary search on a given sorted list and
        # returns the index of the found value, or -1 if not found.
        # Therefore, if a value is in the list it will return a value >= zero.
        #
        # Repeatedly splits the array in half finding a middle point, and setting
        # the middle to to the previous highest value (end) of the array section.
        # If the start becomes greater than or equal to the end, -1 is returned.
        # Since this is recursive, that will be the final return value if the value is not found.
        #
        # PARAMETERS:
        # a0 - array pointer
        # a1 - search value
        # a2 - start index of given section for searching
        # a3 - end index of given section for searching

binarySearch:
#--------------- function beginning  ---------------
        addi    $sp, $sp, -28       # allocate stack space
        sw      $ra, 0($sp)         # store off the return addr, etc
        sw      $s0, 4($sp)
        sw  $s1, 8($sp)
        sw  $s2, 12($sp)
        sw  $s3, 16($sp)
        sw  $s4, 20($sp)
        sw  $s5, 24($sp)
#------------------------- function body -------------------------
    # Place parameters into s registers
    move    $s0, $a0    # s0: array pointer
    move    $s1, $a1    # s1: search value
    move    $s2, $a2    # s2: start index
    move    $s3, $a3    # s3: end index

    bgt $s2, $s3, binarySearchReturnNegative    # if start <= end, then continue, else return -1

    sub $t0, $s3, $s2   # t0: end - start
    #addi   $t1, $s2, 2 # start + 2
    li  $t1, 2
    div $t0, $t1   # (end - start) / 2
    mflo    $s4    # s4: floor of ((end - start) / 2
    add $s4, $s4, $s2   #((end - start) / 2) + start
```

```
    move    $a0, $s4    # a0: mid index
    move    $a1, $s0    # a1: array pointer
    jal getValueAtIndex
    move    $s5, $v0    # s5: array[mid]

    beq $s5, $s1, binarySearchReturnMid         # array[mid] == searchValue
    bgt $s5, $s1, binarySearchReturnRecursiveStart  # return BinarySearch(arr, x, start, mid-1)

    j binarySearchReturnRecursiveEnd # else

binarySearchReturnRecursiveStart:
        move    $a0, $s0 # a0 - array pointer
        move    $a1, $s1 # a1 - search value
        move    $a2, $s2 # a2 - start index of given section for searching
        addi    $t0, $s4, -1 # t0: mid-1
        move    $a3, $t0 # a3 - end index of given section for searching
        jal binarySearch # BinarySearch(arr, x, start, mid-1)
        # RETURN VALUE?
        j binarySearchDone

binarySearchReturnMid:
    move    $v0, $s4    # return mid
    j binarySearchDone

binarySearchReturnRecursiveEnd:
    move    $a0, $s0 # a0 - array pointer
        move    $a1, $s1 # a1 - search value
        addi    $t0, $s4, 1 # t0: mid+1
        move    $a2, $t0 # a2 - start index of given section for searching (mid+1)
        move    $a3, $s3 # a3 - end index of given section for searching
    jal binarySearch # BinarySearch(arr, x, mid + 1, end)
    j binarySearchDone

binarySearchReturnNegative:
    li  $v0, -1 # return -1

binarySearchDone:
#------------------- function end -----------------
        lw      $ra, 0($sp)         # restore the return address, etc
        lw      $s0, 4($sp)
        lw  $s1, 8($sp)
        lw  $s2, 12($sp)
        lw  $s3, 16($sp)
        lw  $s4, 20($sp)
        lw  $s5, 24($sp)
        addi    $sp, $sp, 28
        jr      $ra                 # return to the calling function
#*********************************************************************


#--------------- quickSort Functions


#*********************************************************************
        # quickSort function
        # In-place mutator of given list. No return value.
        #
        # Recursively sorts the array in-place within areas designated by pivots.
    # The average case is O(n log n), though the worst is O(n^2) when given an already sorted list.
    # This function is tied to a helper function, partition(),
    # which simultaneously returns a pivot location for quickSort() and actually sorts the list itself.
        #
        # PARAMETERS:
        # a0 - array to sort
        # a1 - number of values in array
        # a2 - start index of given section for sorting
        # a3 - end index of given section for sorting
```

```
quickSort:
#--------------- function beginning  ---------------
        addi    $sp, $sp, -24        # allocate stack space
        sw      $ra, 0($sp)         # store off the return addr, etc
        sw      $s0, 4($sp)
        sw  $s1, 8($sp)
        sw  $s2, 12($sp)
        sw  $s3, 16($sp)
        sw  $s4, 20($sp)
#------------------------- function body -------------------------
    # Place parameters into s registers
    move    $s0, $a0    # s0: array pointer
    move    $s1, $a1    # s1: length of array
    move    $s2, $a2    # s2: start index
    move    $s3, $a3    # s3: end index

    bgt $s2, $s3, quickSortDone # if start < end, goto function finish. Else:


    # # determine pivot location # #
    move    $a0, $s0    #a0: array pointer
    move    $a1, $s1    #a1: length of array
    move    $a2, $s2    #a2: start
    move    $a3, $s3    #a3: end
    jal quickSortPartition # quickSortPartition(array, arraySize, start, end)
    move    $s4, $v0    # s4: pivot

    # # sort the left side of the array # #
    move    $a0, $s0    #a0: array pointer
    move    $a1, $s1    #a1: length of array
    move    $a2, $s2    #a2: start
    addi    $t0, $s4, -1    # t0: pivot-1
    move    $a3, $t0    # a3: pivot-1
    jal quickSort   # quickSort(array, arraySize, start, pivot-1)


    # # sort the right side of the array # #
    move    $a0, $s0    #a0: array pointer
    move    $a1, $s1    #a1: length of array
    addi    $t0, $s4, 1 # t0: pivot+1
    move    $a2, $t0    # a2: pivot+1
    move    $a3, $s3    #a3: end
    jal quickSort   # # quickSort(array, arraySize, pivot+1, end)

quickSortDone:
#------------------- function end -----------------
        lw      $ra, 0($sp)         # restore the return address, etc
        lw      $s0, 4($sp)
        lw  $s1, 8($sp)
        lw  $s2, 12($sp)
        lw  $s3, 16($sp)
        lw  $s4, 20($sp)
        addi    $sp, $sp, 24
        jr      $ra                 # return to the calling function
#*********************************************************************


#*********************************************************************
        # quickSortPartition function
        # In-place mutator of given list. Returns a pivot value.
        #
        # Mutates the array by rearranging the elements of the given section,
    # so that all elements lesser than the pivot go to the left end of the array,
    # and the ones greater than the pivot go to the right of the array.
    # It returns the pivot value, which grows from the starting end of the section.
        #
```

```
        # PARAMETERS:
        # a0 - array to sort
        # a1 - number of values in array
        # a2 - start index of given section for sorting
        # a3 - end index of given section for sorting
        #
        # RETURN VALUES:
        # v0 - new pivot location

quickSortPartition:
#--------------- function beginning  ---------------
        addi    $sp, $sp, -32       # allocate stack space
        sw      $ra, 0($sp)         # store off the return addr, etc
        sw      $s0, 4($sp)
        sw  $s1, 8($sp)
        sw  $s2, 12($sp)
        sw  $s3, 16($sp)
        sw  $s4, 20($sp)
        sw  $s5, 24($sp)
    sw  $s6, 28($sp)
#------------------------- function body -------------------------
    # Place parameters into s registers
    move    $s0, $a0    # s0: array pointer
    move    $s1, $a1    # s1: length of array
    move    $s2, $a2    # s2: start index
    move    $s3, $a3    # s3: end index


    move    $a0, $s3    # a0: end index
    move    $a1, $s0    # a1: array pointer
    jal getValueAtIndex
    move    $s4, $v0    # s4: pivot = array[end]

    move    $s5, $s2    # s5: left = start
    move    $s6, $s2    # s6: index = start

quickSortPartitionLoop:
    bge $s6, $s3, quickSortPartitionDone # while index < end; for the length of the section

    move    $a0, $s6    # a0: index(s6)
    move    $a1, $s0    # a1: array pointer
    jal getValueAtIndex
    move    $t0, $v0    # t0: array[index]

    bgt $t0, $s4, quickSortPartitionLoopIncrement   # if array[i] > pivot end, aka arr[i] <= pivot then swap

    move $a0, $s6 # a0: index
    move $a1, $s5 # a1: left
    move $a2, $s0 # a2: array
    jal swap # swaps array[left] with array[index]
    addi    $s5, $s5, 1 # left += 1

quickSortPartitionLoopIncrement:
    addi    $s6, $s6, 1 # index += 1
    j quickSortPartitionLoop


quickSortPartitionDone:
    move $a0, $s3 # a0: end
    move $a1, $s5 # a1: left
    move $a2, $s0 # a2: array
    jal swap # swaps array[left] with array[end]
    move    $v0, $s5 # v0: return left
#------------------- function end -----------------
        lw      $ra, 0($sp)         # restore the return address, etc
        lw      $s0, 4($sp)
        lw  $s1, 8($sp)
```

```
        lw  $s2, 12($sp)
        lw  $s3, 16($sp)
        lw  $s4, 20($sp)
        lw  $s5, 24($sp)
        lw  $s6, 28($sp)
        addi    $sp, $sp, 32
        jr      $ra                 # return to the calling function
#*********************************************************************


#------- helper functions


swap:
#--------------- function beginning  ---------------
        addi    $sp, $sp, -4       # allocate stack space for 2 values
        sw      $ra, 0($sp)         # store off the return addr, etc

#------------------------- function body -------------------------
    # t2 - index1 w/ offset
    # t3 - index1 value
    # t4 - index2 w/ offset
    # t5 - index2 value
    li  $t0, 4 # 4 to multiply by

    mul $t1, $a0, $t0 # t1: index1 * 4
    add $t2, $a2, $t1 # t2: load index1 offset (array pointer + index1) into t2
    lw  $t3, 0($t2) # t3: index1 value

    mul $t1, $a1, $t0 # t1: index2 * 4
    add $t4, $a2, $t1 # t4: load index2 (array pointer + index2) into t4
    lw  $t5, 0($t4) # let t5 be index2 value

    # swap
    sw  $t5, 0($t2) # store index2 value into index1
    sw  $t3, 0($t4) # store index1 value into index2

swapDone:
#-------------------- function end -----------------
        lw      $ra, 0($sp)         # restore the return address, etc
        addi    $sp, $sp, 4
        jr      $ra                 # return to the calling function
#*********************************************************************


#*********************************************************************
        # printList function
        #
        # prints a list to output.
        #
        # PARAMETERS:
        # a0 - array pointer
        # a1 - array size pointer

printList:
#--------------- function beginning  ---------------
        addi    $sp, $sp, -20       # allocate stack space for 2 values
        sw      $ra, 0($sp)         # store off the return addr, etc
        sw  $s0, 4($sp)
        sw  $s0, 8($sp)
        sw  $s1, 12($sp)
        sw  $s3, 16($sp)

#------------------------- function body -------------------------
    move    $s0, $0         # s0: index = 0
        move    $s1, $a0            # s1: array pointer: malleable
        move    $s2, $a1        # s2: array size
```

```
        move    $s3, $a0        # s3: array pointer at index 0
printListLoop:
    bge     $s0, $s2, printListDone      # while (index < arraySize)

    move    $a0, $s0    # parameter: currentIndex
    move    $a1, $s3    # parameter:  array pointer
    jal getValueAtIndex

    move    $a0, $v0    #a0: array[currentIndex]
    li      $v0, 1
        syscall # print value



        addi    $s0, $s0, 1         # index++
        addi    $s1, $s1, 4         # update address pointer

        # the below is a bit weird but its a workaround to incrementing the index in only one place
        move    $t0, $s2    # get arraySize temp
        subi    $t0, $t0, 1 # t0: arraySize-1
        ble $s0, $t0, printListArrayComma # if index+1 <= arraySize-1
        j       printListLoop

printListArrayComma: # print comma if not at end of array
    la  $t0, comma  # t0: address to comma
    move    $a0, $t0    # load t0 into syscall parameter
    li  $v0, 4      # call print string
    syscall
    #addi    $s0, $s0, 1         # index++
        #addi    $s1, $s1, 4         # update address pointer
        j printListLoop

printListDone:
#------------------- function end -----------------
        lw      $ra, 0($sp)         # restore the return address, etc
        lw  $s0, 4($sp)
        lw  $s0, 8($sp)
        lw  $s1, 12($sp)
        lw  $s3, 16($sp)
        addi    $sp, $sp, 20
        jr      $ra                 # return to the calling function
#*********************************************************************


#*********************************************************************
        # getValueAtIndex function
        #
        # takes an index and an array pointer,
        # returns the value at array offset.
        #
        # PARAMETERS:
        # a0 - index
        # a1 - array pointer
        # RETURN:
        # v0 - value w/ offset

getValueAtIndex:
#-------------- function beginning  ---------------
        addi    $sp, $sp, -4       # allocate stack space for 2 values
        sw      $ra, 0($sp)         # store off the return addr, etc

#------------------------- function body -------------------------
    sll $a0, $a0, 2     # multiply index by 4
    add $a1, $a1, $a0   # add offset to array pointer
    lw  $v0, 0($a1) # return value at offset location

#------------------- function end -----------------
```

```
        lw      $ra, 0($sp)         # restore the return address
        addi    $sp, $sp, 4
        jr      $ra                 # return to the calling function
#*****************************************************************
```