COL216 Assignment 1 Report

Amaiya Singhal (2021CS50598) Aanya Khurana (2021CS10084)

Overview

The aim of this assignment is to write and implement the MIPS assembly programs for the three specified tasks and analyze their correctness and time complexity. Given below are the input-output specifications, approaches and complexity analysis for each of the three functions.

1. Echo

Input Specifications

An input string prompted by the `asciiz` prompt "Enter input string:"

Output Specifications

The output string printed out as "Output is:"

Approach

- Heap memory is allotted dynamically using `syscall`, and the maximum memory size is specified as 100 bytes as the length of an input string cannot exceed 100 characters.
- The input string is stored at the memory address obtained after allotting heap memory.
- The address of the string is stored in the register $a0, the register $v0 is set to 4 (display mode) and the string is printed out on the console.

Testcases

- Tried on strings of varying sizes containing different characters.
- Tested on empty string and string of size 100.

```
Enter input string: Hello world
Output is: Hello world
```

```
Enter input string: sakjfhcdjnhfclakfcjacflshaclsdhmfcjklsd hklhmcflaksd
Output is: sakjfhcdjnhfclakfcjacflshaclsdhmfcjklsd hklhmcflaksd
```

2. Binary Search

Input Specifications

The length of the input list (n) is prompted by the string "Enter n:", each of the elements of the list (in order) is sequentially prompted by the string, "Enter list elements:" which appears as many times as the specified length of the list, and the element to be searched (x) is prompted by "Enter x:"

Output Specifications

"Yes at index i" if the element is found at index i. "Not found" otherwise

Approach

- Dynamic memory is allotted in the heap, specifying the maximum as 30*4 = 120 bytes as the maximum length of the list is 30 and all elements are integers.
- The sorted array elements are stored one by one in the memory by incrementing the address to store the element by 4 after each element starting from the base address of the array.
- The algorithm performs standard iterative binary search in the array by updating the values of the starting and stopping indices of the search at every iteration, returning the **first occurrence** of the element to be searched.

Correctness and Complexity

- Termination is guaranteed by the condition of breaking out of the loop when start index = stop index. Since after every iteration either start increases or stop decreases, this is guaranteed due to finite n.
- The range is halved at every iteration with a constant number of operations (including conditional branches, logical and arithmetic operations) so the time complexity is O(log n).

Testcases

- Tested on lists of varying sizes from 1 to 30.
- Tested on lists with repeated elements and tried finding index of elements which occur only once, which occur multiple times and those which do not exist in the list.

```
Enter n: 4
Enter list elements: 1
Enter list elements: 2
Enter list elements: 2
Enter list elements: 3
Enter x: 2
Yes at index 1
```

```
Enter n: 19
Enter list elements: 1
Enter list elements: 2
Enter list elements: 2
Enter list elements: 2
Enter list elements: 3
Enter list elements: 3
Enter list elements: 3
Enter list elements: 4
Enter list elements: 4
Enter list elements: 4
Enter list elements: 4
Enter list elements: 4
Enter list elements: 5
Enter list elements: 5
Enter list elements: 6
Enter list elements: 7
Enter list elements: 8
Enter list elements: 8
Enter list elements: 9
Enter x: 4
Yes at index 7
```

3. Fast Exponentiation

Input Specifications:

Two integer inputs: x (base) and n (exponent) prompted by the ASCIIZ prompts "Enter the value of x:" and "Enter the value of n:"

Output Specifications:

An integer printed out with the ASCIIZ prompt "Answer is:"

Approach

- The input x, the input n, and the return address is stored into the stack by the callee.
- The exponent (x^n) is computed recursively by binary exponentiation, wherein at each recursive call, the value of the last bit of n is stored in a temporary register $t0 and x is squared at each call.
- If the last bit is 1 then the product is multiplied by the current value x else product remains as is.
- At the end of the program, when all the bits of n have been exhausted, the callee increments the stack pointer appropriately and jumps to the return address stored in $ra, restoring control to the caller.

Correctness and Complexity

- Termination is guaranteed by the finiteness of n.
- At the end, the exponent of x contains the expanded binary representation of n (For example 3^5 is represented as 3^{4+1}).
- The operations associated with every bit of the exponent (shift right, and, squaring the base) are all O(1) and as there are log n bits, the time complexity is O(log n).

Testcases

- Tested with 1 as the base and exponent.
- Tested with 0 as the base and exponent.
- Tested on various base and exponent combinations.

```
Enter the value of x: 1
Enter the value of n: 0
Answer is: 1
```

```
Enter the value of x: 157
Enter the value of n: 1
Answer is: 157
```

```
Enter the value of x: 0
Enter the value of n: 500
Answer is: 0
```

```
Enter the value of x: 17
Enter the value of n: 5
Answer is: 1419857
```