Prof. Mo Chen
Simon Fraser University

Assignment 3

CMPT 125
Due Feb. 7

This assignment is to be done individually.

Important Note: The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the concepts involved in the questions with other students. If you are in doubt as to what constitutes acceptable discussion, please ask! Further, please take advantage of office hours offered by the instructor and the TA if you are having difficulties with this assignment.

DO NOT:

- Give/receive code or proofs to/from other students
- Use search engines to find solutions for the assignment

DO:

- Meet with other students to discuss assignment (it is best not to take any notes during such meetings, and to re-work assignment on your own)
- Use online resources (e.g. Wikipedia) to understand the concepts needed to solve the assignment

Submission Instructions:

- You may type or write your answer as long as it is readable.
- Submit two files on CourSys
  1. `$SFUID.pdf`, which contains a write-up of your solutions to the assignment
  2. `$SFUID.c`, which contains the code you wrote in Question 2.

Question 1 (25 marks)

Consider the following code, and answer the questions.

```
1  #include<stdio.h>
2
3  void fun(int x) {
4      if(x > 0) {
5          fun(--x);
6          printf("%d\t", x);
7          fun(--x);
8      }
9  }
10
11 int main() {
12     int a = 3;
13     fun(a);
14     return 0;
15 }
```

a) What is the output? (4 marks)

b) Complete the progression of the function call stack for the program in the given table at the end of this document. Each time there is a new function call or a function returns, there should be a depiction of the call stack. This is shown by adding the new function call on top of the previous function calls or removing the function on top when a value is returned by that function. the first coulumn on the left is a depiction of the call stack and every other column must be filled with respect to that. Please notice that every time the stack changes, it must be written in the next row again. (Please refer to page 5 for the call stack table) (21 marks)

Question 2 (25 marks)

The following code is an iterative implementation of binary search.

```
1  // C program to implement iterative Binary Search
2  #include <stdio.h>
3
4  // A iterative binary search function. It returns
5  // the index of x in given array arr[l..r] if found,
6  // and -1 if not
7  int binarySearch(int arr[], int l, int r, int x) {
8      while (l <= r) {
9          int m = l + (r - l) / 2;
10
11         // Check if x is present at mid
12         if (arr[m] == x) {
13             return m;
14         }
15
16         // If x greater, ignore left half
17         if (arr[m] < x) {
18             l = m + 1;
19         }
20
21         // If x is smaller, ignore right half
22         else {
23             r = m - 1;
24         }
25   }
26
27   // if we reach here, then element was
28   // not found
29   return -1;
30 }
31
32 int main(void) {
33     int arr[] = { 2, 3, 4, 10, 40 };
34     int n = sizeof(arr) / sizeof(arr[0]);
35     int x = 10;
36     int result = binarySearch(arr, 0, n - 1, x);
37
38     (result == -1)  ? printf("Element is not found"
```

```
39                                  " in array\n")
40                      : printf("Element is found at "
41                                  "index %d.\n", result);
42      return 0;
43  }
```

a) Rewrite the code using recursion. Use assertions and comments to help the readers understand your code. The code must be uploaded on Coursys. It is important that your code compiles. Note that you must return the *index* of the element if it's found, as stated in the comments on lines 4-6. (10 marks)

b) Compare the two implementations with regard to their running times in terms of big O estimates. (10 marks)

c) Please explain which method (loop or recursion) is better and if it is always the case for every problem? (5 marks)

Table 1: Question 1 part b: funcion call stack table

	Call stack	Last function call (if applicable)	Last function return (if applicable)	Function parameters (for function that is called or returned)	Output value (if application)
1	`main` $a = 3$ $x = a$	`main`	N/A	N/A	N/A

2	`fun`
	$x = 3$
	`main`
	$a = 3$
	$x = 3$
		`fun`	N/A	$x = 3$	N/A

3	`fun`
	$x = 2$
	`fun`
	$x = 3$
	`main`
	$a = 3$
	$x = 3$
		`fun`		$x = 2$	

4	
	
	`fun`
	$x = 3$
	`main`
	$a = 3$
	$x = 3$				

5	
	
	`fun`
	$x = 3$
	`main`
	$a = 3$
	$x = 3$				

	Call stack	Last function call (if applicable)	Last function return (if applicable)	Function parameters (for function that is called or returned)	Output value (if application)
6	`fun` $x = 1$ `fun` $x = 2$ `fun` $x = 3$ `main` $a = 3$ $x = 3$	N/A	`fun`	$x = 0$	0
7	`fun` $x = -1$ `fun` $x = 1$ `fun` $x = 2$ `fun` $x = 3$ `main` $a = 3$ $x = 3$				
8				$x = -1$	
9					

Assignment 3

	Call stack	Last function call (if applicable)	Last function return (if applicable)	Function parameters (for function that is called or returned)	Output value (if application)
10					
11					

12	`fun`
	$x = 3$
	`main`
	$a = 3$
	$x = 3$
				$x = 2$	

13					
14					

Assignment 3

	Call stack	Last function call (if applicable)	Last function return (if applicable)	Function parameters (for function that is called or returned)	Output value (if application)

15	`fun`
	$x = 1$
	`fun`
	$x = 3$
	`main`
	$a = 3$
	$x = 3$				

16	`fun`
	$x = -1$
	`fun`
	$x = 1$
	`fun`
	$x = 3$
	`main`
	$a = 3$
	$x = 3$
		`fun`		$x = -1$	

17					
18					
19					